

Data-Driven Documents



目錄

D3.js API 中文手册

来源：d3 Wiki API 中文手册

> Wiki ▸ **API--**中文手册

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

D3库中所有函数都在“d3”命名空间内。

D3 使用语义版本命名。 你可以使用“d3.version”查看D3的最新版本

D3 API总览

- 行为 - 可重用的交互行为。
- 核心 - 包括选择器，过渡，数据处理，本地化，颜色等。
- 地理 - 球面坐标，经纬度运算。
- 几何 - 提供绘制2D几何图形的实用工具。
- 布局 - 推导定位元素的辅助数据。
- 比例尺 - 数据编码和视觉编码之间转换。
- 可缩放矢量图形 - 提供用于创建可伸缩矢量图形的实用工具。
- 时间 - 解析或格式化时间，计算日历的时间间隔等。

d3 (核心函数)

选择

- d3.event - 访问用于交互的当前用户事件。
- d3.mouse - 获取相对于指定容器的鼠标位置。
- d3.select - 从当前文档中选择一个元素。
- d3.selectAll - 从当前文档中选择多个元素。
- d3.selection - 增强选择器原型，或测试实例类型。
- d3.touch - 获取相对于指定容器的单点触摸位置。
- d3.touches - 获取相对于指定容器的多点触摸位置。
- selection.append - 创建并追加一个新元素。
- selection.attr - 取得或设置属性的值。
- selection.call - 为当前选择调用一个函数。
- selection.classed - 添加或移除CSS类。
- selection.data - 在计算相关的连接时，取得或设置一组元素的数据。
- selection.datum - 取得或设置单个元素的数据，不必计算连接。
- selection.each - 为每个选中的元素调用一个函数。
- selection.empty - 如果选择是空则返回true。
- selection.enter - 为缺失的元素返回占位符。

- selection.exit - 返回不再需要的元素。
- selection.filter - 基于数据过滤选择。
- selection.html - 取得或设置innerHTML内容。
- selection.insert - 在已存在元素之前创建并插入一个元素。
- selection.interrupt - 如果有过渡的话，立即中断当前的过渡。
- selection.node - 返回选择中的第一个节点。
- selection.on - 为交互添加或移除事件监听器。
- selection.order - 重排列文档中的元素，以匹配选择。
- selection.property - 取得或设置行内属性。
- selection.remove - 从当前文档中移除当前元素。
- selection.select - 为每个选中元素的在选择一个后代元素。
- selection.selectAll - 为每个选中元素的在选择多个后代元素。
- selection.size - 返回选择中的元素数。
- selection.sort - 基于数据排列文档中的元素。
- selection.style - 取得或设置样式属性。
- selection.text - 取得或设置文本内容。
- selection.transition - 在选中元素上开启过渡。

过渡

- d3.ease - 自定义过渡时间。
- d3.timer - 开启一段自定义动画定时器。
- d3.interpolate - 插补两个值。
- d3.interpolateArray - 插补两个数组。
- d3.interpolateHcl - 插补两个HCL颜色值。
- d3.interpolateHsl - 插补两个HSL颜色值。
- d3.interpolateLab - 插补两个L*a*b*颜色值。
- d3.interpolateNumber - 插补两个数字值。
- d3.interpolateObject - 插补两个任意对象。
- d3.interpolateRgb - 插补两个RGB颜色值。
- d3.interpolateRound - 插补两个整数。
- d3.interpolateString - 插补两个字符串。
- d3.interpolateTransform - 插补两个2D矩阵变换。
- d3.interpolateZoom - 在两个点之间平滑地缩放平移。
- d3.interpolators - 注册一个自定义的插值器。
- d3.timer.flush - 立即执行一个0延迟的定时器。
- d3.transition - 开启一个动画过渡。
- ease - 一个参数化的缓动函数。
- interpolate - 一个参数化的插值器函数。
- transition.attr - 平滑地过渡到一个新的属性值。

- transition.attrTween - 在两个属性值之间平滑地过渡。
- transition.call - 为当前的过渡调用一个函数。
- transition.delay - 指定每个元素的延迟时间（以毫秒为单位）。
- transition.duration - 指定每个元素的持续时间（以毫秒为单位）。
- transition.each - 为过渡结束时间添加一个监听器。
- transition.ease - 指定一个过渡的缓动函数。
- transition.empty - 如果过渡是空则返回true。
- transition.filter - 基于数据过滤一个过渡。
- transition.node - 返回过渡中的第一个节点。
- transition.remove - 在过渡的最后移除选中的元素。
- transition.select - 为每个选中的元素在一个子元素开启一段过渡。
- transition.selectAll - 为每个选中的元素在多个子元素开启一段过渡。
- transition.size - 返回在选择中元素的数量。
- transition.style - 平滑地过渡到一个新的样式值。
- transition.styleTween - 在两个样式属性值之间平滑地过渡。
- transition.text - 在过渡开始时设置文本内容。
- transition.transition - 当这次过渡结束时，在相同的元素上开启另一段过渡。
- transition.tween - 指定一个自定义的补间操作符作为过渡的一部分运行。

数组

- d3.ascending - 为排序比较两个值。
- d3.bisectLeft - 在排序数组中检索值。
- d3.bisector - 二等分使用访问器或比较器。
- d3.bisectRight - 在排序数组中检索值。
- d3.bisect - 在排序数组中检索值。
- d3.descending - 为排序比较两个值。
- d3.deviation - 计算一组数据的标准差。
- d3.entries - 列出一个关联数组的键值对实体。
- d3.extent - 找出一个数组中的最大值和最小值。
- d3.keys - 列出一个关联数组中的键。
- d3.map - 构建一个新的map。
- d3.max - 找出一个数组中的最大值。
- d3.mean - 计算一组数据的算数平均值。
- d3.median - 计算一组数据的算数中值。
- d3.merge - 合并多个数组为一个数组。
- d3.min - 找出一个数组中的最小值。
- d3.nest - 分层地分组数组元素。
- d3.pairs - 返回一个元素的相邻对数组。
- d3.permute - 按照数组的索引重新排序数组元素。

- d3.quantile - 为一个排好序的数字数组的分位数。
- d3.range - 产生一系列的数值。
- d3.set - 构建一个新的集合。
- d3.shuffle - 随机化一个数组的顺序。
- d3.sum - 计算数字数组的和。
- d3.transpose - 转置一个数组的数组。
- d3.values - 列出关联数组的值。
- d3.variance - 计算数字数组的方差。
- d3.zip - 转置数组的可变数量。
- map.empty - 如果map不包含元素就返回true。
- map.entries - 返回map的实体数组。
- map.forEach - 为每个指定的实体调用一个函数。
- map.get - 为指定的键返回值。
- map.has - 如果map包含指定的值则返回true。
- map.keys - 返回map的键数组。
- map.remove - 为指定的键移除值。
- map.set - 为指定的键设置值。
- map.size - 返回map的实体数量。
- map.values - 返回map的值数组。
- nest.entries - 返回一组键-值元组
- nest.key - 在嵌套层级中添加一个级别。
- nest.map - 返回一个关联数组。
- nest.rollup - 为叶子值指定一个汇总函数。
- nest.sortKeys - 按照键排序叶子嵌套级别。
- nest.sortValues - 按照值排序叶子嵌套级别。
- set.add - 添加指定的值。
- set.empty - 如果集合不含元素的话返回true。
- set.forEach - 为集合中的每个元素调用指定的函数。
- set.has - 如果集合中包含指定值就返回true。
- set.remove - 移除指定的值。
- set.size - 返回集合中的元素数量。
- set.values - 返回集合中的值数组。

数学

- d3.random.bates - 生成具有贝茨分布规律的随机数。
- d3.random.irwinHall - 生成具有Irwin–Hall分布规律的随机数。
- d3.random.logNormal - 生成具有对数正态分布规律的随机数。
- d3.random.normal - 生成具有正态分布规律的随机数。
- d3.transform - 计算2D仿射变换的标准形式。

请求

- d3.csv - 请求一个CSV（逗号分隔值）的文件。
- d3.html - 请求一个HTML文档片段。
- d3.json - 请求一个JSON对象。
- d3.text - 请求一个text文件。
- d3.tsv - 请求一个TSV（制表符分隔值）的文件。
- d3.xhr - 使用XMLHttpRequest请求一个资源。
- d3.xml - 请求一个XML文档片段。
- xhr.abort - 终止未完成的请求。
- xhr.get - 发送一个GET请求。
- xhr.header - 设置一个请求头。
- xhr.mimeType - 设置一个接受请求头并覆盖响应的MIME类型。
- xhr.on - 为“progress”，“load”或“error”事件添加一个事件监听器。
- xhr.post - 发送一个POST请求。
- xhr.response - 设置一个响应映射函数。
- xhr.send - 使用指定的数据和函数发送一个请求。

格式化

- d3.format - 将一个数组格式化为字符串。
- d3.formatPrefix - 为指定的值和精度返回SI 前缀。
- d3.requote - 将字符串转义为正则表达式。
- d3.round - 将值四舍五入到指定小数位。

CSV格式化(d3.csv)

- d3.csv.formatRows - 格式化一组元组为CSV字符串。
- d3.csv.format - 格式化一组对象为CSV字符串。
- d3.csv.parseRows - 解析CSV字符串为元组，忽略首行。
- d3.csv.parse - 把首行数据CSV字符串解析为对象。
- d3.csv - 请求一个CSV文件。
- d3.dsv - 为指定的分隔符和mime类型创建一个解析器/格式化器。
- d3.tsv.formatRows - 格式化一组元组为TSV字符串。
- d3.tsv.format - 格式化一组对象为TSV字符串。
- d3.tsv.parseRows - 解析TSV字符串为元组，忽略首行。
- d3.tsv.parse - 把首行数据TSV字符串解析为对象。
- d3.tsv - 请求一个TSV文件。

本地化

- d3.locale - 使用指定的字符串创建一个本地化。
- locale.numberFormat - 创建一个新的数字格式化器。
- locale.timeFormat - 创建一个新的时间格式化器/解析器。

颜色

- d3.hcl - 指定一种颜色，创建一个HCL颜色对象。
- d3.hsl - 指定一种颜色，创建一个HSL颜色对象。
- d3.lab - 指定一种颜色，创建一个L*a*b*颜色对象。
- d3.rgb - 指定一种颜色，创建一个RGB颜色对象。
- hcl.brighter - 增强颜色的亮度，变化幅度由参数决定。
- hcl.darker - 减弱颜色的亮度，变化幅度由参数决定。
- hcl.rgb - 将HCL颜色对象转化成RGB颜色对象。
- hcl.toString - HCL颜色对象转化为字符串格式。
- hsl.brighter - 增强颜色的亮度，变化幅度由参数决定。
- hsl.darker - 减弱颜色的亮度，变化幅度由参数决定。
- hsl.rgb - 将HSL颜色对象转化成RGB颜色对象。
- hsl.toString - 将HSL颜色对象转化为字符串格式。
- lab.brighter - 增强颜色的亮度，变化幅度由参数决定。
- lab.darker - 减弱颜色的亮度，变化幅度由参数决定。
- lab.rgb - 将L*a*b*颜色对象转化成RGB颜色对象。
- lab.toString - 将L*a*b*颜色对象转化为字符串格式。
- rgb.brighter - 增强颜色的亮度，变化幅度由参数决定。
- rgb.darker - 减弱颜色的亮度，变化幅度由参数决定。
- rgb.hsl - 将RGB颜色对象转化成HSL颜色对象。
- rgb.toString - 将RGB颜色对象转化为字符串格式。

命名空间

- d3.ns.prefix - 访问或扩展已知的XML命名空间。
- d3.ns.qualify - 限定一个前缀名称，例如"xlink:href".

内部

- d3.dispatch - 创建一个定制的事件分发器。
- d3.functor - 创建一个函数并返回一个常量。
- d3.rebind - 重新绑定get/set方法到一个子类上。
- dispatch.on - 注册或者解除注册事件监听器。
- dispatch.type - 为指定的监听器分发事件。

d3.scale (比例尺)

数值比例尺

- d3.scale.identity - 构建一个线性恒等比例尺。
- d3.scale.linear - 构建一个线性比例尺。
- d3.scale.log - 构建一个对数比例尺。
- d3.scale.pow - 构建一个指数比例尺。
- d3.scale.quantile - 构建一个分位数比例尺。
- d3.scale.quantize - 构建一个量化比例尺（值域离散）。
- d3.scale.sqrt - 构建一个平方根比例尺。
- d3.scale.threshold - 构建一个临界值比例尺（值域离散）。
- identity.copy - 复制比例尺。
- identity.domain - 取得或设置比例尺的定义域。
- identity.invert - 等价于恒等函数。
- identity.range - 等价于identity.domain。
- identity.tickFormat - 获取一个用来展示刻度值得格式化器。
- identity.ticks - 取得定义域中典型的值。
- identity - 恒等函数。
- linear.clamp - 启用或者关闭值域的闭合。
- linear.copy - 复制比例尺。
- linear.domain - 取得或设置比例尺的定义域。
- linear.interpolate - 取得或设置输出插值器。
- linear.invert - 取得输出值对应的输入值。
- linear.nice - 扩展比例尺的定义域为一个优化的定义域。
- linear.rangeRound - 设置比例尺的输出范围，并四舍五入。
- linear.range - 取得或设置比例尺的输出范围。
- linear.tickFormat - 获取一个用来展示刻度值得格式化器。
- linear.ticks - 取得定义域中典型的值。
- linear - 取得输入值对应的输出值。
- log.clamp - 启用或者关闭值域的闭合。
- log.copy - 复制比例尺。
- log.domain - 取得或设置比例尺的定义域。
- log.interpolate - 取得或设置输出插值器。
- log.invert - 取得输出值对应的输入值。
- log.nice - 扩展比例尺的定义域为一个优化的10的次方。
- log.rangeRound - 设置比例尺的输出范围，并四舍五入。
- log.range - 取得或设置比例尺的输出范围。
- log.tickFormat - 获取一个用来展示刻度值得格式化器。
- log.ticks - 取得定义域中典型的值。

- log - 取得输入值对应的输出值。
- pow.clamp - 启用或者关闭值域的闭合。
- pow.copy - 复制比例尺。
- pow.domain - 取得或设置比例尺的定义域。
- pow.exponent - 取得或设置指数。
- pow.interpolate - 取得或设置输出插值器。
- pow.invert - 取得输出值对应的输入值。
- pow.nice - 扩展比例尺的定义域为一个优化的定义域。
- pow.rangeRound - 设置scale的输出范围，并四舍五入。
- pow.range - 取得或设置比例尺的值域。
- pow.tickFormat - 获取一个用来展示刻度值得格式化器。
- pow.ticks - 取得定义域中典型的值。
- pow - 取得输出值对应的输入值。
- quantile.copy - 复制比例尺。
- quantile.domain - 取得或设置比例尺的定义域（离散的值）。
- quantile.invertExtent - 取得输出值对应的输入值。
- quantile.quantiles - 取得比例尺的分位数箱阈值。
- quantile.range - 取得或设置比例尺的值域（离散的值）。
- quantile - 取得输入值对应的输出值。
- quantize.copy - 复制比例尺。
- quantize.domain - 取得或设置比例尺的定义域。
- quantize.invertExtent - 取得输出值对应的输入值。
- quantize.range - 取得或设置比例尺的值域（离散的值）。
- quantize - 取得输入值对应的输出值。
- threshold.copy - 复制比例尺。
- threshold.domain - 取得或设置比例尺的定义域。
- threshold.invertExtent - 取得输出值对应的输入值。
- threshold.range - 取得或设置比例尺的值域（离散的值）。
- threshold - 取得输入值对应的输出值。

序数比例尺

- d3.scale.ordinal - 构造一个序数比例尺。
- ordinal - 获取输入值对应的输出值。
- ordinal.domain - 获取或指定比例尺的输入域。
- ordinal.range - 获取或指定比例尺的输出范围。
- [ordinal.rangePoints] (序数比例尺.md#ordinal_rangePoints) - 指定输出范围为连续区间。
- ordinal.rangeRoundPoints - 指定输出范围为连续区间，刻度点均为整数。
- ordinal.rangeBands - 指定输出范围为连续区间。

- ordinal.rangeRoundBands - 指定输出范围为连续区间，区间段的起点均为整数。
- [ordinal.rangeBand] (序数比例尺.md#ordinal_rangeBand) - 获取区间段的宽度。
- [ordinal.rangeExtent] (序数比例尺.md#ordinal_rangeExtent) - 获取当前比例尺的输出范围，未被切分的。
- ordinal.copy - 深度拷贝当前比例尺对象。
- d3.scale.category10 - 构造一个有10种颜色的序数比例尺。
- d3.scale.category20 - 构造一个有20种颜色的序数比例尺。
- d3.scale.category20b - 构造一个另外20种颜色的序数比例尺。
- d3.scale.category20c - 构造一个另外20种颜色的序数比例尺。

SVG函数

形状

- arc.centroid - 计算弧中心。
- arc.cornerRadius - 获取或设置拐角（corner）半径访问器。
- arc.endAngle - 获取或设置结束角度访问器。
- arc.innerRadius - 获取或设置内半径访问器。
- arc.outerRadius - 获取或设置外半径访问器。
- arc.padAngle - 获取或设置填补（pad）角度访问器。
- arc.padRadius - 获取或设置填补（pad）半径访问器。
- arc.startAngle - 获取或设置开始角度访问器。
- arc - 生成一个像饼图或圆环图中的固定弧度。
- area.angle - 获取或设置角度访问器。
- area.defined - 控制面积在给定点是否是有定义的。
- area.defined - 控制径向面积在给定点是否是有定义的。
- area.endAngle - 获取或设置角度（顶线）访问器。
- area.innerRadius - 获取或设置内半径（基线）访问器。
- area.interpolate - 获取或设置插值模式。
- area.outerRadius - 获取或设置外半径（顶线）访问器。
- area.radius - 获取或设置半径访问器。
- area.startAngle - 获取或设置角度（基线）访问器。
- area.tension - 获取或设置基本样条线的张力。
- area.x0 - 获取或设置*x0*-坐标（基线）访问器。
- area.x1 - 获取或设置*x1*-坐标（顶线）访问器。
- area.x - 获取或设置*x*-坐标访问器。
- area.y0 - 获取或设置*y0*-坐标（基线）访问器。
- area.y1 - 获取或设置*y1*-坐标（顶线）访问器。
- area.y - 获取或设置*y*-坐标访问器。
- area - 生成一个像面积图中的分段线性面积。

- area - 生成一个像极坐标面积图中的分段线性面积。
- chord.endAngle - 获取或设置圆弧结束角访问器。
- chord.radius - 获取或设置圆弧半径访问器。
- chord.source - 获取或设置圆弧来源圆弧访问器。
- chord.startAngle - 获取或设置圆弧开始角访问器。
- chord.target - 获取或设置目标圆弧访问器。
- chord - 在弦图中生成一个二次贝塞尔曲线连接两个弧。
- d3.svg.arc - 新建一个弧度生成器。
- d3.svg.area.radial - 新建一个径向面积生成器。
- d3.svg.area - 新建一个面积生成器。
- d3.svg.chord - 新建一个弦生成器。
- d3.svg.diagonal.radial - 新建一个径向对角线生成器。
- d3.svg.diagonal - 新建一个对角线生成器。
- d3.svg.line.radial - 新建一个径向线生成器。
- d3.svg.line - 新建一个线生成器。
- d3.svg.symbolTypes - 一组符号类型。
- d3.svg.symbol - 新建一个符号生成器。
- diagonal.projection - 设置或获取一个可选的点转换。
- diagonal.source - 设置或获取源点访问器。
- diagonal.target - 设置或获取目标点访问器。
- diagonal - 生成一个像节点链接图中的二维贝塞尔连接器。
- diagonal - 生成一个像节点链接图中的二维径向贝塞尔连接器。
- line.angle - 设置或获取角度 accessor.
- line.defined - 控制线在给定点是否是定义的。
- line.defined - 控制径向线在给定点是否是定义的。
- line.interpolate - 设置或获取插值模式。
- line.interpolate - 设置或获取径向弦的插值模式。
- line.radius - 设置或获取半径访问器。
- line.tension - 设置或获取径向基本样条线的张力。
- line.tension - 设置或获取基本样条线的张力。
- line.x - 设置或获取*x*-坐标访问器。
- line.y - 设置或获取*y*-坐标访问器。
- line - 生成一个像线图中的分段线段。
- line - 生成一个像极线图中的分段线段。
- symbol.size - 设置或获取符号尺寸（平方像素）访问器。
- symbol.type - 设置或获取符号类型访问器。
- symbol - 生成一个像散点图中的符号。

轴

- axis.innerTickSize - 指定内刻度大小。
- axis.orient - 设置或者取得轴的方向。
- axis.outerTickSize - 指定外刻度大小。
- axis.scale - 设置或者取得比例尺。
- axis.tickFormat - 重载标签的刻度格式化。
- axis.tickPadding - 指定刻度和刻度标签之间的间距。
- axis.tickSize - 指定主要的次要的和尾部的刻度。
- axis.ticks - 控制轴的刻度如何生成。
- axis.tickValues - 明确地指定刻度值。
- axis - 为给定的选择器或过渡创建或者更新轴。
- d3.svg.axis - 创建一个新的轴生成器。

刷子

- brush.clear - 重置拖选范围。
- brush.empty - 拖选是否为空。
- brush.event - 在设置范围之后分发拖选事件。
- brush.extent - 拖选范围可以是0,1，2维的。
- brush.on - 监听拖选何时改变。
- brush.x - 拖选的*x*-比例，用于水平拖选。
- brush.y - 拖选的*y*-比例，用于垂直拖选。
- brush - 将拖选应用在指定的选择器和过渡上。
- d3.svg.brush - 点击和拖曳来选择一个1维或2维区域。

d3.time (时间)

时间格式化

- d3.time.format.iso - ISO 8601 UTC时间格式化器。
- d3.time.format.multi - 创建一个新的本地多功能时间格式化器。
- d3.time.format.utc - 由指定的限定符创建一个新的UTC时间格式化器。
- d3.time.format - 由指定的限定符创建一个新的本地时间格式化器。
- format.parse - 将字符串解析为时间对象。
- format - 将一个时间对象格式化为一个字符串。

时间比例尺

- d3.time.scale - 构造一个线性时间比例尺。
- scale.clamp - 指定输出范围是否闭合。
- scale.copy - 创建比例尺的副本。

- scale.domain - 取得或设置比例尺度的定义域。
- scale.interpolate - 取得或设置比例尺的输出插值器。
- scale.invert - 取得给定输出值对应定义域中的值。
- scale.nice - 扩展比例尺的定义域为一个优化的整数值。
- scale.rangeRound - 设置比例尺的四舍五入输出范围。
- scale.range - 取得或设置比例尺的输出范围。
- scale.tickFormat - 取得用于展示刻度值的格式化器。
- scale.ticks - 取得定义域中有代表性的值。
- scale - 取得给定定义域中值对应的输出范围中的值。

时间间隔

- d3.time.dayOfYear - 计算天数。
- d3.time.days - day.range的别名。
- d3.time.day - 每天（12:00 AM）。
- d3.time.fridayOfYear - 计算基于周五的星期数。
- d3.time.fridays -friday.range的别名。
- d3.time.friday - 每周五（例如February 5, 12:00 AM）。
- d3.time.hours - hour.range的别名。
- d3.time.hour - 每个小时（例如, 1:00 AM）。
- d3.time.interval - 一个基于本地时间的时间间隔。
- d3.time.minutes - minute.range的别名。
- d3.time.minute - 每分钟（例如, 1:02 AM）。
- d3.time.mondayOfYear - 计算基于周一的星期数。
- d3.time.mondays - monday.range的别名。
- d3.time.monday - 每周一（例如, February 5, 12:00 AM）
- d3.time.months - month.range的别名。
- d3.time.month - 每个月（例如, February 1, 12:00 AM）
- d3.time.saturdayOfYear - 计算基于周六的星期数。
- d3.time.saturdays - saturday.range的别名。
- d3.time.saturday - every Saturday （例如, February 5, 12:00 AM）。
- d3.time.seconds - second.range的别名。
- d3.time.second - 每秒（例如, 1:02:03 AM）。
- d3.time.sundayOfYear - 计算基于周日的星期数。
- d3.time.sundays - sunday.range的别名。
- d3.time.sunday - 每周日（例如February 5, 12:00 AM）。
- d3.time.thursdayOfYear - 计算基于周四的星期数。
- d3.time.thursdays - thursday.range的别名。
- d3.time.thursday - 每周四（例如February 5, 12:00 AM）。
- d3.time.tuesdayOfYear - 计算基于周二的星期数。

- d3.time.tuesdays - tuesday.range的别名。
- d3.time.tuesday - 每周二（例如February 5, 12:00 AM）。
- d3.time.wednesdayOfYear - 计算基于周三的星期数。
- d3.time.wednesdays - wednesday.range的别名。
- d3.time.wednesday - 每周三（例如February 5, 12:00 AM）。
- d3.time.weekOfYear - sundayOfYear的别名。
- d3.time.weeks - sunday.range的别名。
- d3.time.week - sunday的别名。
- d3.time.years - year.range的别名。
- d3.time.year - 每年（例如January 1, 12:00 AM）。
- interval.ceil - 上取整到最近的时间间隔。
- interval.floor - 下取整到最近的时间间隔。
- interval.offset - 基于一些间隔返回时间偏移。
- interval.range - 返回指定范围中的日期。
- interval.round - 四舍五入到最近的时间间隔。
- interval.utc - 返回UTC时间间隔。
- interval - interval.floor的别名。

d3.layout (布局)

捆布局

- bundle - 对边使用Holten 层次捆绑 算法。
- d3.layout.bundle - 构造一个新的默认的捆绑布局。

弦布局

- chord.chords - 取回计算的弦角度。
- chord.groups - 取回计算的分组角度。
- chord.matrix - 取得或设置布局需要的矩阵数据。
- chord.padding - 取得或设置弦片段间的角填充。
- chord.sortChords - 取得或设置用于弦的比较器（Z轴顺序）。
- chord.sortGroups - 取得或设置用于分组的比较器。
- chord.sortSubgroups - 取得或设置用于子分组的比较器。
- d3.layout.chord - 从关系矩阵生成一个弦图。

簇布局

- cluster.children - 取得或者设置子节点的访问器函数。
- cluster.links - 技术树节点之间的父子连接。

- cluster.nodeSize - 为每个节点指定固定的尺寸。
- cluster.nodes - 计算簇布局并返回节点数组。
- cluster.separation - 取得或设置邻接节点的分隔函数。
- cluster.size - 取得或设置布局的尺寸。
- cluster.sort - 取得或设置兄弟节点的比较器函数。
- cluster - cluster.nodes的别名。
- d3.layout.cluster - 将实体聚集成树状图。

力布局

- d3.layout.force - 使用物理模拟排放链接节点的位置。
- force.alpha - 取得或者设置力布局的冷却参数。
- force.chargeDistance - 取得或者设置最大电荷距离。
- force.charge - 取得或者设置电荷强度。
- force.drag - 给节点绑定拖动行为。
- force.friction - 取得或者设置摩擦系数。
- force.gravity - 取得或者设置重力强度。
- force.linkDistance - 取得或者设置链接距离。
- force.linkStrength - 取得或者设置链接强度。
- force.links - 取得或者设置节点间的链接数组。
- force.nodes - 取得或者设置布局的节点数组。
- force.on - 监听在计算布局位置时的更新。
- force.resume - 重新加热冷却参数，并重启模拟。
- force.size - 取得或者设置布局大小。
- force.start - 当节点变化时启动或者重启模拟。
- force.stop - 立即停止模拟。
- force.theta - 取得或者设置电荷作用的精度。
- force.tick - 运行布局模拟的一步。

层次布局

- d3.layout.hierarchy - 派生一个定制的层次布局实现。
- hierarchy.children -取得或设置子节点的访问器。
- hierarchy.links - 计算树节点中的父子链接。
- hierarchy.nodes - 计算层次布局并返回节点数组。
- hierarchy.revalue - 重新计算层次值。
- hierarchy.sort - 取得或设置兄弟节点的比较器函数。
- hierarchy.value - 取得或设置值访问器函数。
- hierarchy - hierarchy.nodes的别名。

直方图布局

- d3.layout.histogram - 构造一个新的默认的直方图布局。
- histogram.bins - 指定值是如何组织到箱中的。
- histogram.frequency - 按频数或者频率计算分布。
- histogram.range - 取得或设置值得范围。
- histogram.value - 取得或设置值访问器。
- histogram - 使用量化的箱计算数据的分布。

包布局

- d3.layout.pack - 用递归的圆-包生成一个层次布局。
- pack.children - 取得或设置子节点的访问器。
- pack.links - 计算树节点中的父子链接。
- pack.nodes - 计算包布局并返回节点数组。
- pack.padding - 指定布局间距（以像素为单位）
- pack.radius - 指定节点半径（不是由值派生来的）
- pack.size - 指定布局尺寸。
- pack.sort - 控制兄弟节点的遍历顺序。
- pack.value - 取得或设置用于圆尺寸的值访问器。
- pack - pack.nodes的别名。

分区布局

- d3.layout.partition - 递归地将节点树分区为旭日图或者冰柱图。
- partition.children - 取得或设置孩子访问器。
- partition.links - 计算树节点中的父子链接。
- partition.nodes - 计算分区布局并返回节点数组。
- partition.size - 指定布局的尺寸。
- partition.sort - 控制兄弟节点的遍历顺序。
- partition.value - 取得或设置用来指定圆尺寸的值访问器。
- partition - partition.nodes的别名。

饼布局

- d3.layout.pie - 构造一个新的默认的饼布局。
- pie.endAngle -取得或设置饼布局整体的结束角度。
- pie.padAngle - 取得或设置饼布局填充角度。
- pie.sort - 控制饼片段的顺时针方向的顺序。
- pie.startAngle - 取得或设置饼布局整体的开始角度。
- pie.value - 取得或设置值访问器函数。

- pie - 计算饼图或圆环图中弧的开始和结束角度。

堆叠布局

- d3.layout.stack - 构造一个新的默认的堆叠布局。
- stack.offset - 指定整体的基线算法。
- stack.order - 控制每个系列的顺序。
- stack.out - 取得或设置用于存储基线的输出函数。
- stack.values - 取得或设置每个系列的值访问器函数。
- stack.x - 取得或设置*x*-维访问器函数。
- stack.y - 取得或设置*y*-维访问器函数。
- stack - 计算堆叠图或者面积图的基线。

树布局

- d3.layout.tree - 整齐地排列树节点。
- tree.children - 取得或设置孩子访问器。
- tree.links - 计算树节点的父-子连接。
- tree.nodeSize - 为每个节点指定一个固定的尺寸。
- tree.nodes - 计算父布局并返回一组节点。
- tree.separation - 取得或设置相邻节点的间隔函数。
- tree.size - 用*x*和*y*指定树的尺寸。
- tree.sort - 控制遍历顺序中兄弟节点的顺序。
- tree - tree.nodes的别名。

矩形树布局

- d3.layout.treemap - 使用空间递归分区算法展示树的节点。
- treemap.children - 取得或设置孩子访问器。
- treemap.links - 计算树节点中的父子链接。
- treemap.mode - 改变布局的算法。
- treemap.nodes - 计算矩形树布局并返回节点数组。
- treemap.padding - 指定父子之间的间距。
- treemap.round - 启用或者禁用四舍五入像素值。
- treemap.size - 指定布局的尺寸。
- treemap.sort - 控制兄弟节点的遍历顺序。
- treemap.sticky - 让布局对稳定的更新是粘滞的（sticky）。
- treemap.value - 取得或设置用来指定矩形树中矩形单元尺寸的值访问器。
- treemap - treemap.nodes的别名。

d3.geo (地理)

地理路径

- circle.angle - 指定角半径（以度为单位）。
- circle.origin - 指定经纬度原点。
- circle.precision - 指定分段圆的精度。
- circle - 生成一个分段圆。
- d3.geo.area - 计算给定要素的球体面积。
- d3.geo.bounds - 计算给定要素的经纬度边界框。
- d3.geo.centroid - 计算给定要素的球体中心。
- d3.geo.circle - 创建一个圆生成器。
- d3.geo.distance - 计算两点之间的大弧距离。
- d3.geo.graticule - 创建一个经纬网生成器。
- d3.geo.interpolate - 两个点之间插入一个大弧。
- d3.geo.length - 计算线的长度或多边形的面积。
- d3.geo.path - 创建一个地理路径生成器。
- d3.geo.rotation - 为指定的角度[λ, φ, γ]创建一个旋转角度。
- graticule.extent - 取得或设置major & minor范围。
- graticule.lines - 为经线和纬线生成线数组。
- graticule.majorExtent - 取得或设置major范围。
- graticule.majorStep - 取得或设置major步长间隔。
- graticule.minorExtent - 取得或设置minor范围。
- graticule.minorStep - 取得或设置minor步长间隔。
- graticule.outline - 生成格子线范围的一个多边形。
- graticule.precision - 取得或设置纬度精度。
- graticule.step - 取得或设置major & minor步长间隔。
- graticule - 生成经纬线的多线要素。
- path.area - 计算给定要素的投影面积。
- path.bounds - 计算给定要素的投影边界。
- path.centroid - 计算给定要素的投影中心。
- path.context - 取得或设置渲染上下文。
- path.pointRadius - 取得或设置点要素的半径。
- path.projection - 取得或设置地理投影。
- path - 投影指定的要素并渲染上下文。
- rotation.invert - 反旋转球体周围的给定位置。
- rotation - 旋转球体周围的给定位置。

地理投影

- albers.parallels - 取得或者设置投影的两条标准平行线。
- d3.geo.albersUsa - 用于展示美国地图的Albers复合投影。
- d3.geo.albers - Albers等面积圆锥投影。
- d3.geo.azimuthalEqualArea.raw - 原始方位角等面积投影。
- d3.geo.azimuthalEqualArea - 方位角等面积投影。
- d3.geo.azimuthalEquidistant.raw - 原始方位角等距投影。
- d3.geo.azimuthalEquidistant - 方位角等距投影。
- d3.geo.conicConformal.raw - 原始圆锥正形投影。
- d3.geo.conicConformal - 圆锥正形投影。
- d3.geo.conicEqualArea.raw 原始圆锥等面积投影 (a.k.a. Albers)。
- d3.geo.conicEqualArea 圆锥等面积投影 (a.k.a. Albers)。
- d3.geo.conicEquidistant.raw - 原始圆锥等距投影。
- d3.geo.conicEquidistant - 圆锥等距投影。
- d3.geo.equirectangular.raw - 原始等角投影（普通圆柱投影）。
- d3.geo.equirectangular - 等角投影（普通圆柱投影）。
- d3.geo.gnomonic.raw - 原始球心投影。
- d3.geo.gnomonic - 球心投影。
- d3.geo.mercator.raw - 原始墨卡托投影。
- d3.geo.mercator - 球形墨卡托投影。
- d3.geo.orthographic.raw - 原始方位角直角投影。
- d3.geo.orthographic - 方位角直角投影。
- d3.geo.projectionMutator - 从可变的原始投影创建一个标准投影。
- d3.geo.projection - 从原始投影创建一个标准投影。
- d3.geo.stereographic.raw - 原始方位角立体投影。
- d3.geo.stereographic - 方位角立体投影。
- d3.geo.transverseMercator.raw - 原始横向墨卡托投影。
- projection.center - 取得或设置投影的中心位置。
- projection.clipAngle - get or set the radius of the projection's clip circle.
- projection.clipExtent - 取得或设置投影的视口剪切范围（以像素为单位）。
- projection.invert - 为指定的位置反转投影。
- projection.precision - 取得或设置自适应重采样的精度阈值。
- projection.rotate - 取得或设置投影的三轴旋转角。
- projection.scale - 取得或设置投影的缩放系数。
- projection.stream - 包装指定的流监听器，投影输入的几何图形。
- projection.translate - 取得或设置投影的平移位置。
- projection - 投影指定的位置。

流

- clipExtent.extent - 设置剪裁范围。

- d3.geo.clipExtent - 流转换剪切几何图形为给定的轴对齐矩形。
- d3.geo.stream - 将GeoJSON对象转换为几何流。
- d3.geo.transform - 转换流几何图形。
- stream.lineEnd - 表示线或者环的终点。
- stream.lineStart - 表示线或者环的起点。
- stream.point - 表面*x*, *y* (可选的 *z*) 坐标。
- stream.polygonEnd - 表明多边形的终点。
- stream.polygonStart - 表明多边形的起点。
- stream.sphere - 表明一个球体。
- transform.stream - 包装指定的流。

d3.geom (几何)

泰森多边形

- d3.geom.voronoi - 用默认的访问器创建一个泰森多边形布局。
- voronoi.clipExtent -取得或者设置铺嵌的剪切范围。
- voronoi.links - 计算Delaunay mesh为一个链接网络。
- voronoi.triangles - 计算Delaunay mesh为一个三角形密铺。
- voronoi.x - 取得或者设置每个点的x-坐标访问器。
- voronoi.y - 取得或者设置每个点的y-坐标访问器。
- voronoi - 为每个指定的点计算泰森多边形密铺。

四叉树

- d3.geom.quadtree - 为一个点数组创建一个四叉树。
- quadtree.add - 添加点到四叉树中。
- quadtree.find - 找到四叉树中最近的点。
- quadtree.visit - 递归地遍历四叉树中的点。

多边形

- d3.geom.polygon - 由指定的点数组创建多边形。
- polygon.area - 计算多边形逆时针方向的面积。
- polygon.centroid - 计算多边形的面积中心。
- polygon.clip - 对这个多边形进行执行的多边形剪切。

凸包

- d3.geom.hull - 使用默认访问器创建一个convex hull布局。

- hull - 为给定的点数组计算convex hull。
- hull.x - 取得或设置*x*-坐标访问器。
- hull.y - 取得或设置*y*-坐标访问器。

d3.behavior (行为)

拖动

- d3.behavior.drag - 创建拖动行为。
- drag.on - 监听拖动事件。
- drag.origin - 设置拖动行为的原点。

缩放

- d3.behavior.zoom - 创建缩放行为。
- zoom.center - 鼠标滚轮缩放的焦点。
- zoom.duration - 取得或设置双击事件的过渡持续的时间。
- zoom.event - 设置完缩放比例或平移之后分发缩放事件。
- zoom.on - 事件监听器。
- zoom.scaleExtent - 可选参数，比例因子范围。
- zoom.scale - 当前的比例因子。
- zoom.size - 视口的大小。
- zoom.translate - 当前的平移偏移量。
- zoom.x - 可选比例尺，其定义域绑定到视口的*x*范围。
- zoom.y - 可选比例尺，其定义域绑定到视口的*y*范围。
- zoom - 给指定的元素应用缩放行为。

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]]

具体如下:

- [[选择器]] - 操作当前文档中的元素。
- [[过渡]] - 平滑地插入属性和样式。
- [[数组]] - 方便地操作数组和对象。
- [[请求]] - 加载外部数据。
- [[格式化]] - 转换数字，时间和其他对象为字符串。
- [[本地化]] - 控制具有本地特征的行为，例如数字格式化。
- [[颜色]] - 解析和操作颜色；使用颜色空间。
- [[命名空间]] - 扩展D3对XML命名空间的支持。
- [[数学]] - 数学函数杂项。
- [[内部]] - 各种用于扩展D3的工具。

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 选择器

- guluT20141102
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

一个选择就是从当前文档中抽取的一组元素。D3使用[[CSS3|http://www.w3.org/TR/css3-selectors/]]来选择页面元素。例如，你可以使用的选择方式有标签 ("div")、类(".awesome")、唯一标识符("#foo")、属性("[color=red]")、或者包含("parent child")。选择器可以是交叉(".this.that" 表示逻辑与)的也可以是联合的(".this, .that" 表示逻辑或)。如果你的浏览器不支持选择器，你可以在D3前包含[[Sizzle|http://sizzlejs.com/]]来支持向后兼容。

当选择完元素之后你可以添加操作。这些操作可以设置或者获取属性、样式、特性、选择器和文本内容。属性值等可以被指定为常量或者函数；后者为每个元素求值。你也可以为数据选择器加入到data操作；这个数据就可以用做数据驱动的变换的操作符。另外，加上数据产生enter和exit子选择器，你就可以add和remove元素来响应数据中的改变。

一般来说，在D3中你不需要使用for循环后者递归来修改文档。因为你可以一次操作这个选择器而不是遍历各个元素。当然，你也可以手动地遍历各个元素，有一个each操作可以调用任意一个函数，选择器的结果是数组，所以元素可以访问(例如 `selection[0][0]`)。为了简便D3支持方法链：这个操作返回值是一个选择器。

选择元素

D3提供了两种高级方法来选择元素：select和selectAll。这些方法接收选择器字符串。前者只返回第一个匹配的元素，后者选择在文档遍历次序中所有匹配的元素。这个方法也可以接受节点，这可以用来和第三方库例如JQuery或者开发工具(`$0`)整合。

d3.**select**(*selector*)

选中与指定选择器字符串匹配的第一个元素，返回单元素选择结果。如果当前文档中没有匹配的元素则返回空的选择。如果有多个元素被选中，只有第一个匹配的元素（在文档遍历次序中）被选中。

d3.**select**(*node*)

选择指定的节点。如果你已经有一个节点的引用这将是很有用的。例如事件监听器中的 `d3.select(this)` 或者一个全局对象例如document.body`。这个函数不会遍历DOM树。

d3.**selectAll**(*selector*)

选中匹配指定选择器的所有的元素。这些元素会按照文档的遍历顺序（从上到下）选择。如果当前文档中没有匹配的元素则返回空的选择。

d3.**selectAll**(*nodes*)

选择指定的元素数组。如果你已经有了一些元素的引用这将是很有用的，例如事件监听器中的 `d3.selectAll(this.childNodes)` ，或者一个全局对象例如 `document.links` 。节点的参数不用恰好是数组；任何可以转化为一个数组的伪数组（例一个 `NodeList` 或者 arguments`）都可以，这个函数不会遍历DOM树。

操作选择

选择是一组元素。D3绑定额外的方法到数组上。所以你可以在选中的元素上应用操作，例如为所有选中的元素设置属性值。一个细微的差别是选择结果是分组的，而不是一个一维数组。每一个选择都是元素数组中的一个数组。这保留了自选择的层次结构，大多数情况下你可以忽略这个细节，这就是为什么一个单一的元素选择看起来像 `[[node]]` 而不是 `[node]` 。想要了解更多关于嵌套选择的知识请参考嵌套选择。如果你想要学习选择器是怎么工作的你可以用浏览器的开发者工具控制台交互式地访问数据。你可以检查返回数组来查看哪些元素被选中了以及它们是如何分组的。你还可以再应用操作来选择元素并查看页面内容的改变。

内容

D3有一系列可以影响文档内容的操作。这些是你最常用来展示数据的。当操作用来设置文档内容的时候会返回当前选择，所以你就可以使用一个简单的声明将多个操作链接在一起。

selection.**attr**(*name*[, *value*])

如果指定了*value*参数，将为所有选中的元素通过指定的*name*为指定的*value*设置属性。如果*value*是一个常数，那么所有的元素都将设置为相同的属性值；如果*value*是一个函数，那么这个函数会为每个选中的元素（按顺序）计算。入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。这个函数的返回值接下来用来设置每个元素的属性。null值将移除指定的属性。

如果*value*参数没有指定，就会返回为选择中第一个非空（null）元素所指定的属性值。一般来说，只有当你知道选择中恰好包含一个元素时才有用。

指定的*name*参数也会含有一个前缀，例如 `xlink:href` 是用来指定Xlink命名空间中*href*属性的。默认情况下，D3支持svg、xhtml、xlink、xml和 xmlns命名空间。可以添加d3.ns.prefix来注册其他的命名空间。

*name*也可以是一个*name*和*value*属性对象。

selection.**classed**(*name*[, *value*])

这个操作是用来设置*class*属性值得便捷程序。它能识别*class*属性是一个按照空格分隔的标记集合。这样它就能使用[[classList|https://developer.mozilla.org/en/DOM/element.classList]]（如果有的话）来方便地添加、移除和切换CSS类。

如果*value*属性被指定，不论是否指定类都会与选定元素相结合。如果*value*是一个常量且其值为真，那么所有的元素都会被分配指定的类（还没分配的话）。如果其值为假，那么就会移除选中元素的class（已经分配过）。如果*value*是一个函数，那么这个函数会为每个选中的元素（按顺序）计算。入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。这个函数的返回值接下来用来分配或者不分配每个元素的class。

如果你想一次设置多个class可以使用一个对象，文字如同： `selection.classed({'foo': true, 'bar': false})` ，或者使用以空格分隔的class列表形如： `selection.classed('foo bar', true)` 。

如果*value*没有被指定，当且仅当选择中首个非空值有指定的class就会返回true。一般来说，只有当你知道选择中恰好包含一个元素时才有用。

selection.**style**(*name*[, *value*[, *priority*]])

如果*value*参数被指定，通过指定名称和指定的值为所有选中的元素设置CSS样式属性。如果*value*是一个常数，那么所有的元素都设置相同的样式值；否则，如果值是一个函数，则该函数为每个选定的元件（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用来设置每个元素的样式属性。 null值将删除样式属性。可选参数*priority*也可以指定，无论是null空或字符串“important”（不带感叹号）。

如果你想一次设置多个样式属性，使用对象文本，如下所示：

```
selection.style({'stroke': 'black', 'stroke-width': 2})
```

如果未指定值，则返回在选择中的第一个非空元素指定样式属性的当前计算值。只有当你知道选择只包含一个元素时是很有用的。需要注意的是计算的值可能与先前设置的值不同，尤其是当样式属性使用了简写属性（如“font”样式，这是简写为"font-size"，"font-face",“等）。

selection.**property**(*name*[, *value*])

一些HTML元素具有特殊的属性使用标准的属性或样式是不可寻址的。例如，表单文本（text）字段有一个 `value` 字符串属性，复选框（checkboxes）有一个 `checked` 布尔型属性。可以使用 `property` 操作符来获取或设置这些属性，，或任何其他基本元素的可寻址字段，例如 `className` 。

如果指定了*value*，就为所有选中的元素指定名称的属性设置指定的值（value）。如果值是一个常数，那么所有的元素被给予相同的属性值；如果*value*是一个函数，则该函数为每个选定的元素（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i` ，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用于设置每个元素的属性。空值将删除指定的属性。

如果你想一次设置多个属性，可以使用对象文本，如下所示：`selection.property({'foo': 'bar', 'baz': 'qux'})`。

如果未指定值，则返回在选择中第一个非空元素指定属性的值。只有当你知道选择只包含一个元素这通常是很有用的。

selection.**text**([*value*])

文本操作符是基于 `textContent` 属性；设置文本内容将取代任何现有的子元素。

如果指定了*value*时，设置所有选择元素的文本内容为指定的值。如果*value*是一个常数，那么所有的元素被赋予相同的文本内容；如果*value*是一个函数，则该函数被每个选定的元素（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i`，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用于设置每个元素的文本内容。null值会清除内容。

如果未指定*value*，则返回在选择中第一个非空元素的文本内容。只有当你知道选择只包含一个元素时这通常是很有用的。

selection.**html**([*value*])

`html` 的操作基于[[innerHTML|http://dev.w3.org/html5/spec-LC/apis-in-html-documents.html#innerhtml]]属性；设置内部HTML内容将取代任何现有的子元素。此外，您可能更愿意使用数据驱动的方式`append`或`insert`操作创建HTML内容;该操作符的目的是，适用于你想用少量但有丰富格式的HTML。

如果指定了*value*，为所有选择的元素设置在内部的HTML内容为指定的值。如果*value*是一个常数，那么所有的元素被给予相同的内部HTML内容；如果*value*是一个函数，则该函数为每个选定的元素（按顺序）计算，入参是当前数据元素 `d` 和当前索引 `i`，以及代表当前DOM元素的 `this` 上下文。该函数的返回值被用于设置每个单元的内部的HTML内容。null值会清除内容。

如果未指定*value*，则返回在选择中第一个非空元素的内部HTML内容。只有当你知道选择只包含一个元素时这通常是很有用的。 注：正如它的名字所暗示的，selection.html仅支持HTML元素。 SVG元素和其它非HTML元素不支持innerHTML属性，因此与selection.html不相容。请考虑使用XMLSerializer转换DOM树为文本。灵参见innersvg polyfill，它提供了一个垫片以支持SVG元素的innerHTML属性。

selection.**append**(*name*)

在当前选择的每个元素最后追加具有指定名称的新元素，返回包含追加元素的新选择。每个新的元素继承当前元素的数据（如果有的话）和select相同的方式使用子选择。

这个*name*可以被指定为一个常量字符串或一个函数，返回追加的DOM元素。当name被指定为一个字符串，它可能有以下形式的命名空间前缀“namespace:tag”。例如，“svg:text”将在svg命名空间创建“text”元素。默认情况下，D3支持svg，xhtml，xlink的，xml和xmlns命名空

间。其他的命名空间可以通过添加到d3.ns.prefix注册。如果没有指定命名空间，那么命名空间会从封闭的元素继承；或者，如果该名称是已知的前缀之一，相应的命名空间将被使用（例如，“svg”表示“svg:svg”）。

selection.**insert**(*name*[, *before*])

在当前选择与指定before选择器匹配的每个元素之前插入具有指定*name*的新元素，返回包含插入的元素的一个新的选择。如果before选择器不匹配任何元素，那么新元素将用append追加为最后一个子元素。每一个新元素继承当前元素（如果有的话）的数据，子选择（subselections）和select以同样的方式。

这个*name*可以被指定为一个常量字符串或一个函数，返回追加的DOM元素。当*name*被指定为一个字符串，它可能有以下形式的命名空间前缀“namespace:tag”。例如，“svg:text”将在svg命名空间创建“text”元素。默认情况下，D3支持svg，xhtml，xlink的，xml和xmlns命名空间。其他的命名空间可以通过添加到d3.ns.prefix注册。如果没有指定命名空间，那么命名空间会从封闭的元素继承；或者，如果该名称是已知的前缀之一，相应的命名空间将被使用（例如，“svg”表示“svg:svg”）。

同样地，*before*选择器可以被指定为一个选择器字符串或一个函数，它返回一个DOM元素。例如，`insert("div", ":first-child")`将在当前选择前面加上div子节点。对于enter选择器，*before*选择器在这种情况下也可以省略：输入的元素将被立即插入到更新选择紧随的兄弟元素前（如果有的话）。这使您可以插入DOM的元素与绑定的数据是一致的顺序。但是请注意，如果更新元素修改了顺序，selection.order可能仍然需要。

selection.**remove**()

删除从当前文档当前选择中的元素。返回“屏幕外（off-screen）”的当前选择（除去了相同的元素），从DOM分离。需要注意的是目前还没有一个专门的API来重新添加删除的元素到文档；然而，你可以通过一个函数来selection.append或selection.insert重新添加元素。

数据

selection.**data**([*values*[, *key*]])

连接指定的一组数据的和当前选择。指定的*values*是一组数据值（例如，数字或对象）或一个函数返回一组值。如果没有指定*key*函数，则*values*的第一数据被分配到当前选择中第一元素，第二数据分配给当前选择的第二个元素，依此类推。当数据被分配给一个元素，它被存储在属性`__data__`中，从而使数据“沾粘”，从而使数据可以再选择。

`data`操作的结果是*update*选择;这表示选择的DOM元素已成功绑定到指定的数据元素。*update*选择还包含对enter和exit的选择，对应于添加和删除数据节点。有关详细信息，请参阅简短的教程关于连接的思考

*key*函数可以被指定为控制数据是如何连接元素。这取代默认的by-index行为;*key*函数被新数据数组中的每个元素调用一次，并再次用于选择中的每个元素。在这两种情况下的*key*函数是通过传递数据d与索引i。当key 函数上被新的数据元素评价时， `this` 上下文是数据数组；当key 函数被现有选择评估时， `this` 上下文是相关的DOM元素。*key*函数，基于先前结合的数据返回一个用于连接数据和相关的元素的字符串。例如，如果每个数据都有一个唯一的字段 `name` ，该连接可以被指定为 `.data(data, function(d) { return d.name; })` 。如果指定了*key*函数， `data` 操作符也影响节点的索引；该索引被作为第二个参数 `i` 作为任何运算符函数的参数。然而，请注意，现有的DOM元素不自动重新排序；根据需要使用sort或order函数。有关*key*函数如何影响数据连接的更详细的示例，请参阅教程[[条形图2|http://bost.ocks.org/mike/bar/2/]]

这个*values*选择中的每组数据。因此，如果选择具有多个组（例如，一个d3.selectAll后跟一个selection.selectAll)），然后data应该被指定为一个函数，该函数返回一个数组（假设你对每个组想要不同的数据）。该函数将被传递的当前组数据（或 `undefined` ）和索引，组的 `this` 上下文。 例如，可以将一个二维数组和初始选择绑定，然后将包含的内部数组和每个子选择绑定。在这种情况下，*values*函数是标识函数：它被每个组中的子元素调用，被传递绑定到父元素的数据，并且返回这个数据数组。

```
var matrix = [
  [11975,  5871, 8916, 2868],
  [ 1951, 10048, 2060, 6171],
  [ 8010, 16145, 8090, 8045],
  [ 1013,   990,  940, 6907]
];

var tr = d3.select("body").append("table").selectAll("tr")
    .data(matrix)
  .enter().append("tr");

var td = tr.selectAll("td")
    .data(function(d) { return d; })
  .enter().append("td")
    .text(function(d) { return d; });
```

如果未指定*values*，则此方法返回选择中的第一组数据的数组。返回的数组的长度，将与第一组的长度匹配，并且在返回的数组中的每个数据的索引将匹配选择中相应的索引。如果选择的某些元素为null，或者如果他们有没有相关的数据，则数组中的相应元素将是 `undefined` 。

注意： `data` 方法不能用于清除先前结合数据；可以使用selection.datum代替。

selection.**enter()**

返回输入（enter）选择：当前选择中存在但是当前DOM元素中还不存在的每个数据元素的占位符节点。此方法只在由data运算符返回的更新选择中定义。此外，输入选择只定义了append（append），insert（insert），select（select）和call（call）操作符；您必须使用这些操作符在修改任何内容之前实例化输入元素。当你传递函数的参数给这些插入的元素的操作符时，index参数将反映新的位置，而不一定从零开始或者是连续的。 （输入选择也支持empty和size。）

举一个简单的例子，考虑现有的选择是空的情况下，我们希望创建新的节点来匹配我们的数据：

```
d3.select("body").selectAll("div")
    .data([4, 8, 15, 16, 23, 42])
  .enter().append("div")
    .text(function(d) { return d; });
```

假设body最初是空的，上面的代码将创建六个新的div元素，将它们按顺序追加到body，并指定其文本内容为相应的（强制转为字符串）号码：

```
<div>4</div>
<div>8</div>
<div>15</div>
<div>16</div>
<div>23</div>
<div>42</div>
```

另一种方式考虑进入的占位符的节点是，它们是指向父节点（在该示例中，就是文档的body）；然而，他们只支持追加和插入。 当你追加或插入时输入选择并入到更新选择。而不是对enter和update选择单独采用同样的操作符，现在你可以一次添加节点后，将其应用到更新选择。如果你发现自己移除整个选择的元素才重新插入其中大部分，用这个来替代。例如：

```
var update_sel = svg.selectAll("circle").data(data)
update_sel.attr(/* operate on old elements only */)
update_sel.enter().append("circle").attr(/* operate on new elements only */)
update_sel.attr(/* operate on old and new elements */)
update_sel.exit().remove() /* complete the enter-update-exit pattern */
```

selection.**exit()**

返回退出（exit）选择：找出在当前选择存在的DOM元素中没有新的数据元素时。此方法只被定义在data运算符返回的更新选择。exit选择定义了所有的正常操作符，但通常你主要使用的是remove；其他操作符存在的主要目的是让您可以根据需要定义一个退出的过渡。请注意，exit操作符只是返回一个exit选择引用，由你来删除新节点。 一个简单的例子，考虑更新在上面的例子中的enter操作符创建的六个DIV元素。在这里，我们把这些元素和一个含有一些新的，一些旧数据的新数组绑定：

```
var div = d3.select("body").selectAll("div")
    .data([1, 2, 4, 8, 16, 32], function(d) { return d; });
```

现在div--data操作符的结果--指的是更新的选择。因为我们指定的key函数使用标识函数，并且将新数据数组包含数字[4,8,16]，它也存在旧的数据数组中，这个更新选择包含3 个DIV元素。比方说，我们离开这些元素原样。我们可以实例化并使用enter选择添加新的元素[1,2,32]：

```
div.enter().append("div")
    .text(function(d) { return d; });
```

同样，我们可以删除退出的元素[15, 23, 42]： div.exit().remove();

```
现在，文档body如下：
<div>4</div>
<div>8</div>
<div>16</div>
<div>1</div>
<div>2</div>
<div>32</div>
```

注意，DOM元素现在是乱序。然而，选择索引i（操作函数的第二个参数），将正确地与新数据数组相匹配。例如，我们可以指定一个索引属性：

```
d3.selectAll("div").attr("index", function(d, i) { return i; });
```

结果是：

```
<div index="2">4</div>
<div index="3">8</div>
<div index="4">16</div>
<div index="0">1</div>
<div index="1">2</div>
<div index="5">32</div>
```

如果你想在文档遍历，以匹配选择数据顺序，可以使用sort或者order。

selection.**filter**(*selector*)

过滤选择，返回一个新的选择只包含其指定的*selector*是true的元素。*selector*可以被指定为一个函数或作为选择的字符串，如“.foo”。和其他操作符一样，该函数被传递当前数据 `d` 和索引 `i`，以及this上下文作为当前的DOM元素。过滤器应该只在选择与DOM元素绑定的时候。例如：从append或insert。只绑定的元素和数据的一个子集，可以在data参数中调用内置的数组过滤器filter。像内置函数一样，D3的过滤器不会在返回选择中保留原来的选择的索引；返回移除元素的副本。如果您想保留的索引，使用select代替。 例如，要选择奇数索引（相对于从零开始的索引）的每一个元素：

```
var odds = selection.select(function(d, i) { return i & 1 ? this : null; });
```

等价地，使用的过滤器函数：

```
var odds = selection.filter(function(d, i) { return i & 1; });
```

或过滤选择器（注意：nth-child伪类是一开始的索引，而不是从零开始的索引）：

```
var odds = selection.filter(":nth-child(even)");
```

因此，您可以使用选择或过滤器操作符应用到元素的一个子集上。

selection.**datum**([*value*])

获取或设置每个选定的元素绑定的数据。不像selection.data方法，这种方法不计算一个连接（并因此不计算enter和exit的选择）。此方法在selection.property之上实现：

```
d3.selection.prototype.datum = function(value) {
  return arguments.length < 1
      ? this.property("__data__")
      : this.property("__data__", value);
};
```

如果指定*value*，就为所有选中的元素设置元素的绑定数据为指定的值。如果*value*是一个常数，所有的元素被给予相同的数据；否则，如果*value*是一个函数，则该函数为每个选定的元素计算，被传递以前的数据 `d` 与当前索引 `i`，使用 `this` 上下文作为当前的DOM元素。该函数之后被用来确定每个元素的数据。null值将删除绑定的数据。该操作数对索引没有影响。

如果未指定*value*，则返回在选择中绑定第一个非空的元素的数据。只有当你知道选择只包含一个元素这通常是很有用的。

注意：此方法是以前所谓的“map”。旧名已被弃用。

datum 方法用D3访问HTML5自定义数据属性非常有用。例如，给定下列元素：

```
<ul id="list">
  <li data-username="shawnbot">Shawn Allen</li>
  <li data-username="mbostock">Mike Bostock</li>
</ul>
```

你可以公开通过设置每个元素的数据作为内置的dataset属性自定义数据属性D3：

```
selection.datum(function() { return this.dataset; })
```

这可以用来，例如，按照用户名排序元素。示例

selection.**sort**([*comparator*])

根据指定的*comparator*函数对当前选择的元素排序。比较器函数默认为d3.ascending，通过传递两个数据元素*a*和*b*进行比较，并返回任一负的，正的，或零值。如果为负，则*a*应该在*b*之前；如果为正，则*a*应该为*b*后；否则a和b被认为是相等的顺序是任意的。需要注意的是，这种排序是不保证是稳定的；然而，它保证与浏览器内置的数组排序方法具有相同的行为。

selection.**order**()

重新插入元素到文档这样文档顺序选与择顺序就相匹配。这等同于调用sort()如果数据已经排序，但要快得多。

动画和交互

selection.**on**(*type*[, *listener*[, *capture*]])

在当前选择的每个元素，为指定的类型*type*，添加或删除事件监听器*listener*。*type*是一个字符串事件类型的名称，如“点击”，“鼠标悬停”，或“提交”。基本上支持任何DOM事件。有关D3支持的事件类型的更多详细信息，请查看Mozilla或Stackoverflow。指定的listener 与其他操作符函数调用方式相同，被传递的当前数据 `d` 和索引 `i` 与 `this` 上下文作为当前的DOM元素。为了在侦听器内访问当前事件，使用全局函数d3.event。事件侦听器的返回值将被忽略。

如果所选择的元素相同类型的一个事件监听已经注册了，新的侦听加入之前的现有侦听被除去。为注册相同事件类型的多个监听器，该类型可以跟一个可选的命名空间，如“click.foo”和“click.bar”。 要删除一个监听器，传递null给*listener*，删除特定事件类型所有监听，传递null给listener，指定 `.type` 的类型，如：`selection.on(".foo", null)`。

一个可选的捕获*capture*标志可以指定，对应于W3C的useCapture标志：“开始捕获后，所有指定类型的事件将被分派到注册的EventListener在被分派到事件树任何EventTargets下，事件是沿着树向上冒泡而不会触发一个EventListener指定使用捕获”。

如果未指定监听器，指定类型（如果有）返回当前分配的监听器。

d3.**event**

存储当前的事件（如果有的话）。这个全局函数是在事件侦听器回调过程中使用on操作符注册的。当监听器在 `finally` 块被通知后当前事件被重置。这使侦听器函数与其他操作符函数具有相同的形式，传递当前数据 `d` 和索引 `i`。 `d3.event` 对象是DOM事件，并实现了标准事件字段，像时间戳 `timeStamp` 和键代码 `keyCode`，以及 `preventDefault()` 方法和的 `stopPropagation()` 方法。当然你可以使用原生事件的pageX and pageY，它往往更方便转变事件位置为接收该事件容器的局部坐标系。例如，如果你在网页的正常流程嵌入SVG，你可能想事件的位置是相对于SVG图像的左上角的。如果您的SVG包含转换，你也可能想知道事件的相对于那些变换的位置。标准鼠标指针使用d3.mouse运算符，d3.touches用在iOS多点触控事件上。

d3.**mouse**(*container*)

返回当前d3.event相对于指定的容器的x和y坐标，。该容器可以是一个HTML或SVG容器元素，如svg:g或svg:svg。该坐标返回为一个包含两个元素的数组[*x*, *y*]。

d3.**touch**(*container*[, *touches*], *identifier*)

返回指定标识符触摸当前[d3.event](#)相应的x和y坐标，相对于指定容器。如果未指定*touches*时，默认为当前事件的[changedTouches](#)。该容器可以是一个HTML或SVG容器元件，如一个的svg:g或svg:svg。坐标被返回为两元素数组的数组[[*x1*, *y1*], [*x2*, *y2*], …]。如果在指定的标识符*touches*没有接触，则返回null；这对于忽略touchmove那些只是touches移动了的事件是很有用的。

d3.**touches**(*container*[, *touches*])

返回与当前[d3.event](#)相关联的每个触摸*x*和*y*坐标，基于[touches](#)属性，相对于指定的容器中。该容器可以是一个HTML或SVG容器元素，如svg:g 或 svg:svg。坐标返回两个元素的数组的数组[[*x1*, *y1*], [*x2*, *y2*], …]。如果指定了触摸，返回指定触摸的位置，如果没有指定*touches*时，则默认为当前事件的 `touches` 属性。

selection.**transition**([*name*])

开始为当前选择的[过渡](#)。转换的行为很像选择，除了操作符动画平滑的随着时间的推移，而不是瞬间完成。

selection.**interrupt**([*name*])

立即中断当前的[过渡](#)（如果有的话）。不会取消任何尚未启动的预定的转变。最好要取消原定过渡，简单地创建中断当前的过渡后，一个新的零延迟过渡：

```
selection
    .interrupt() // 取消当前过渡
    .transition(); //抢占任何预定的转换
```

子选择

顶层的select方法查询整个文档，一个选择的[select](#)和[selectAll](#)操作符限定查询每个选定元素的后代；我们称之为“部分选择”。例如， `d3.selectAll("p").select("b")` 返回在每一个段落("p")元素中的第一个bold("b")元素。部分选择通过selectAll通过祖先分组元素。因此， `d3.selectAll("p").selectAll("b")` 。通过段落分组而d3.selectAll("p b") 返回一个扁平的选择。部分选择通过select是相似的，但保留了团体和传播数据。分组起在数据连接中起重要的作用，和功能性操作符可依赖于其组内的当前元素的数值索引。

selection.**select**(*selector*)

对当前选中的每个元素，选中第一个匹配特定的选择器字符串*selector*的子代元素。如果当前元素没有元素匹配特定的选择器，当前索引处的元素在返回的选择中将是空值null。运算符（除了[data](#)）自动跳过null元素，从而保持现有选择的索引。如果当前元素具有相关联的数据，该数据由返回的子选择继承，并且自动绑定到新选定的元素。如果有多个元素匹配选择器，只有在文档遍历顺序中第一个匹配的元素会被选中。选择器也可以被指定为一个函数，

返回一个元素，或者null（如果没有匹配的元素）。在这种情况下，指定的选择器以和其他操作函数同样的方式被调用，被传递的当前数据 `d` 和索引 `i` ，与 `this` 上下文作为当前的DOM元素。

selection.**selectAll**(*selector*)

对当前选择的每个元素，选取匹配指定选择器字符串的后代元素。返回的选择是通过在当前选择的祖先节点进行分组。如果没有元素和当前元素的指定选择器相匹配，返回的选择中当前索引处的组将是null。部分选取不从当前选择继承数据，但如果数据值指定为一个函数，这个函数会被调用，带有的祖先节点的数据d和组索引i来确定部分选定的数据绑定。

通过selectAll的分组也将影响后续进入的占位节点。因此，为追加进入的节点指定父节点，使用select后紧跟selectAll：

```
d3.select("body").selectAll("div")
```

您可以通过检查各组数据的parentNode属性查看每个组的父节点，例如selection[0].parentNode。

选择器也可以被指定为一个函数，返回元素的数组（或节点列表NodeList），或者空数组（如果没有匹配的元素）。在这种情况下，指定的选择器以和其他操作函数同样的方式被调用，被传递的当前数据 `d` 和索引 `i` ，与 `this` 上下文作为当前的DOM元素。

控制

对于高级用法，D3有一些额外的用户控制流操作符。

selection.**each**(*function*)

为当前选择的每个元素，调用指定的函数，传递当前数据 `d` 和索引 `i` ，与当前的DOM元素的 `this` 上下文。这个操作符几乎被所有的其他操作符内部使用，并可以用于调用任意代码为每个选定的元素。 `each` 操作符可以用来处理递归的选择，通过在回调函数内使用 `d3.select(this)` 。

selection.**call**(*function*[, *arguments…*])

调用指定的函数一次，通过在当前的选择以及任何可选参数。无论指定函数的返回值是什么，call操作符总是返回当前的选择。通过call调用函数与手动调用函数是完全一样的；但它可以更容易地使用方法链。例如，假设我们要在许多不同的地方以同样的方式设置一些属性。我们采取的代码，把它包在一个可重复使用的功能：

```
function foo(selection) {
  selection
      .attr("name1", "value1")
      .attr("name2", "value2");
}
```

现在我们可以这样写：

```
foo(d3.selectAll("div"));
```

或者等价的方式：

```
d3.selectAll("div").call(foo);
```

被调用函数的 `this` 上下文也是当前的选择。第一个参数是略显多余的，这我们可能在未来解决。 如果您使用的 `selection.call` 的对象，方法和需要 `this` 指向该对象创建之前调用绑定到对象的函数。

```
function Foo(text) {
    this.text = text;
}

Foo.prototype.setText = function(selection) {
    selection.text(this.text);
}

var bar = new Foo("Bar");

d3.selectAll("span").call(bar.setText.bind(bar));
// Or
d3.selectAll("span").call(Foo.prototype.setText.bind(bar));
```

selection.**empty**()

返回true如果当前选择为空；一个选择是空的，如果它不包含任何元素或null元素。

selection.**node**()

返回当前选择的第一个非空的元素。如果选择为空，则返回null。

selection.**size**()

Returns the total number of elements in the current selection.

扩展

d3.**selection**()

返回根的选择，相当于 `d3.select(document.documentElement)` 。此函数还可以用于检查一个对象是一个选择： `o instanceof d3.selection` 。您还可以添加新的方法到选择的原形中。例如，添加一个便捷的方法，用于设置复选框的 `checked` 属性，你可能这样写：

```js d3.selection.prototype.checked = function(value) { return arguments.length < 1 ? this.property("checked") : this.property("checked", value); };
```

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 过渡

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

过渡是一种特殊类型的选择器([[selection|选择器]])，这种操作符的应用随时间平滑而不是瞬间变换。你可以使用过渡[[transition()|选择器#transition]] 操作符从选择得到一个过渡。但是，过渡通常支持和选择器（例如attr和style）一样的操作符，不是所有的操作符都支持。例如，你必须在过渡开始前添加元素。移除remove操作符用来在过渡结束后方便地移除元素。

过渡可能有每个元素的延迟和持续时间，使用类似于其他操作符的数据计算功能。这可以很容易地给不同元素切换过渡，不论基于数据还是索引。例如，你可以给元素排序然后在变换中交错过渡得到更好的元素重排序效果。想了解更多关于此技术的内容请参考Heer和Robertson的Animated Transitions in Statistical Data Graphics。

D3有许多内置插值来简化任意值的过渡。例如，你可以从字体字符串“500 12px sans-serif”到“300 42px sans-serif”过渡，D3将发现嵌入在字符串内的数字自动地插入字体尺寸（size）和重量（weight）。您甚至可以插入任意的嵌套对象和数组或SVG路径数据。如果你找到内置插值的不足，D3允许使用attrTween和styleTween操作符自定义的内插器。D3的插值器提供[[比例尺|比例尺]]的基础并且可以用在过渡的外部。内插器是一个函数将定义域[0,1]中的而一个值t映射为一个颜色（color），数字或者任意的值。

更多关于过渡的内容可以参考Working with Transitions 教程。

对给定的元素在给定的时间只有一个过渡被激活。然而，在相同的元素可能安排多个过渡；只要是错开的时间，每一个过渡将按顺序运行。如果有新转变给定元素上运行，它隐含取消所有旧的转变，包括任何被安排但尚未运行。这允许一个新的过渡，例如那些将对一个新的用户事件响应的过渡，将会取代老的过渡，即使那些旧的过渡已经上演过了或者延迟上演了。多级过渡（过渡是在较早一个过渡的end事件中创建的过渡）被认为是和原始的过渡相同的“年龄”的；这是内部通过在单调增加的唯一ID跟踪的，其中ID是创建多级过渡时继承的。

开始过渡

\# d3.**transition**([*selection*], [*name*])

创建动画过渡。这等价于 `d3.select(document).transition()` 。这个方法很少使用，因为它通常更容易从现有选择导出的转变，而不是从现有的过渡导出的选择。当使用可选的选择调用，这个方法通常返回指定选择；例如它是一个空操作。然而，transition.each的上下文中，这个方法将为指定的选择（继承了延迟，持续和其他父过渡中的属性）创建一个新的过渡。这用来实现可以不论在选择器或者在过渡中都能被调用的可重用组件是非常有用的。在后者的情况下支持导出并行过渡。相关的一个实例可以参考D3的axis component。

transition.**delay**([*delay*])

指定过渡延迟（*delay*）以毫秒为单位。如果延迟是一个常量，则所有的元素将被赋予相同的延迟；如果延迟是一个函数，则这个函数将被每个选中的元素（按顺序）计算，将被传递当前的数据d和当前的索引i作为函数的变量，使用this上下文作为当前DOM元素。这个函数的返回值将被用来为每个元素的延迟设置值。默认的延迟是0。如果*[[延迟(duration)|过渡#duration]]*没有被指定，就返回过渡中第一个非空元素绑定的delay值。设定延迟为索引i的倍数是一种方便方式错开元素过渡。例如，如果你使用duration 中固定的持续，并且当前选择有n个元素，你可以通过2 * *duration* 错开过渡：

```
.delay(function(d, i) { return i / n * duration; })
```

你也可以计算出延迟作为数据的一个功能，从而产生一个数据驱动的动画。

注意: 延迟总是相对一系列过渡中的第一个而言.

transition.**duration**([*duration*])

指定每个元素的持续时间（duration），单位为毫秒。如果持续时间是常量的，那么所有的元素被给予相同的持续时间；否则，如果持续时间是函数，则该函数为每个选定的元件（按顺序）计算持续时间，被传递的当前数据d与当前索引i，并用this表示当前的DOM元素。该函数的返回值被用于设置每个元素的持续时间。默认持续时间为250毫秒。如果没有指定时间，则返回绑定过渡（transition）中的第一个非空元素的持续时间。

transition.**ease**([*value*[, *arguments*]])

指定过渡的缓动函数(easing function。如果*value*参数是一个函数，它被用于缓解当前参数定时值*t*，*t*通常在范围[0,1]内。（在过渡结束时，*t*可以是稍大于1）否则，*value*参数被假定为一个字符串，这个参数被传递给[[d3.ease|Transitions#d3_ease]]方法来产生一个缓动函数。默认缓动函数是“cubic-in-out”。需要注意的是它不能用来定制每个元件或每个属性的缓动函数；但是，如果你使用的“线性（linear）”缓和函数，可以通过使用attrTween或styleTween插值器中内置的定制缓动功能。如果未指定缓动，则返回绑定到此过渡（transition）中的第一个非空元素的缓动函数。

Operating on Transitions

Content

transition.**attr**(*name*, *value*)

通过指定的*name*和*value*过渡属性的值。过渡的初始值是当前属性值（一定要事先设定一个初始值，如果你不想坏的意外），结束值是指定的值。如果value 是常量的，那么所有的元素被给予相同的属性值；否则，如果value 是函数，则该函数为每个选定的元件（按顺序）计算

属性值，被传递的参数是当前数据d与当前索引i，用this表示当前的DOM元素。该函数的返回值被用于设置每个元素的属性。不支持null值因为插值器将变成未定义的（undefined）；如果你想在过渡结束后移除属性值可以使用remove函数。 内插器基于终值自动选择。如果终值是一个数字，开始值被强制转换为一个数字，并使用interpolateNumber内插器。如果结束值是一个字符串，执行一个检查，以查看该字符串是否代表一个颜色形如/^(#|rgb(|hsl()/或CSS命名颜色中的一个；如果是，则初始值被强制为RGB颜色，使用interpolateRgb内插器。否则，interpolateString内插器被使用，在字符串中插入数字。

transition.**attrTween**(*name*, *tween*)

根据指定的补间（tween）函数，通过指定的名称（name）过渡属性值。过渡的开始和结束值由补间函数决定。补间函数在每个元素的过渡开始的时候被调用，传入参数有当前数据d，当前索引i，this上下文表示当前DOM元素。补间函数的返回值必须是一个插值器：函数将定义域[0,1]映射为颜色、数字或者属性值。例如，attr操作符是建立在attrTween操作符之上的。补间函数被attr操作符使用取决于终值是一个函数还是常量。如果终值是一个函数则：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value.call(this, d, i)));
}
```

Otherwise, if the end value is a constant: 如果终值是一个常量则：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value));
}
```

这个attrTween操作符在你需要一个自定义的内插器的时候用到，例如：理解SVG路径数据语义。一种常见的技术是数据空间插值：其中interpolateObject用来插入两个数据值，这个插值的结果（也就是用一个shap）用来计算新属性值。使用attr操作符处理简单常见的情况，即内插器可以自动地从当前属性值转到期望的终值。

transition.**style**(*name*, *value*[, *priority*])

通过指定的名称（name）和值（value）过渡CSS样式属性值。可选参数优先级（priority）也可以指定为null或者字符串"important"（不带感叹号）。过渡的起始值是当前计算的样式属性值，终值是指定的value。如果value是一个常量那么所有的元素都被指定为相同的样式属性值；否则，如果value是一个函数那么函数将为每个选定的元素（按顺序）计算值，被传递的参数有当前数据d，当前索引i，this上下文代表当前DOM元素。函数的返回值之后被用来转换每个元素的样式属性值。null是不支持的，因为插值器需要是undefined；如果你想在转换之后移除样式属性，参考end事件。 内插器基于终值自动选择。如果终值是一个数字，开始值被强制转换为一个数字，并使用interpolateNumber内插器。如果结束值是一个字符串，执行一个检查，以查看该字符串是否代表一个颜色形如/^(#|rgb(|hsl()/或CSS命名颜色中的一个；如果是，则初始值被强制为RGB颜色，使用interpolateRgb内插器。否则，interpolateString内插器被使用，在字符串中插入数字。 需要注意的是所计算的初始值也可以和以前设定的值不

同，尤其是当样式属性使用的缩写属性（如“font”样式，简写为“font-size”，“font-face”，等等）。此外，计算尺寸，如“font-size”和“line-height”是以像素为单位，所以你也应该适当地指定以像素为单位的终值。

transition.**styleTween**(*name*, *tween*[, *priority*])

根据指定的补间函数指定的名称过渡CSS样式属性的值。可选参数priority 可以被指定为null或者字符串“important”（没有感叹号）。过渡的起始和结束值由补间函数决定；补间函数在没给元素过渡开始的时候调用，被传递的参数是当前数据d，当前索引I，当前属性值a，this上下文代表当前DOM元素。补间函数的返回值必须是一个插值器：一个函数映射定义域[0,1]中的值t为一个颜色、数字或者属性值。 例如，style操作符是建立在styleTween操作符之上的。补间函数被style操作符使用取决于终值是一个函数还是常量。终值是函数时：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value.call(this, d, i)));
}
```

终值是常量时：

```
function tween(d, i, a) {
    return d3.interpolate(a, String(value));
}
```

styleTween操作符在需要定制插值器的使用使用到，例如：理解CSS3转换的语义。对简单常用的情况使用style操作符，即插值器可以从当前计算的样式属性自动驱动到期望的终值。

transition.**text**(*value*)

文本（text）操作符是基于textContent 属性：设置文本内容将取代任何现有的子元素。在转换开始时设置所有选定元素的文本内容为指定值。如果值是一个常数，那么所有的元素被给予相同的文本内容；否则，如果值是一个函数，则该函数被每个选定的元件（按顺序）计算，被传递当前数据d与当前索引i，this上下文作为当前的DOM元素。该函数的返回值被用于设置每个元素的文本内容。null值会清除内容。

transition.**tween**(*name*, *factory*)

注册一个自定义补间指定的名称（name）。当在过渡开始时，指定的工厂函数将被过渡中每个选定的元素调用，传递该元素的数据（d）和索引（i）作为参数，元素作为上下文（this）。工厂应反回补间函数，在整个过渡过程中被调用。补间函数之后被反复调用，传递当前的归一化时间t在[0，1]范围内。如果工厂返回null，则补间不会在选定元素上运行。 补间函数是用来内部实现attr和style补间，并可以用来对其它文档内容进行内插。例如，内插从0到100的文本内容：

```
selection.transition().tween("text", function() {
    var i = d3.interpolateRound(0, 100);
    return function(t) {
        this.textContent = i(t);
    };
});
```

补间常使用闭包捕捉过渡开始时创建的状态。上例中，i在过渡开始的时候初始化，随后用在过渡过程中（虽然注意到，在上面的例子中，转变的开始值是硬编码为0，而过渡更常用的初始值是基于DOM中的当前状态）。 # transition.**remove**()

在过渡结束时删除选定的元素。如果在任何选定的元素有新的过渡计划，这些元素不会被删除；然而，“end”事件仍会被调度。 Subtransitions 过渡可以从现有的过渡派生，类似subselections的方式。 Subtransitions继承缓动，持续和延迟父过渡。

transition.**select**(*selector*)

当前过渡的每个元素中，选择指定的选择字符串匹配的第一个后代元素。如果当前元素没有元素匹配指定的选择器字符串，那么返回的选择中当前索引的元素将是null。操作符（除数据）自动的跳过空值，从而保持现有选择的索引。如果当前元素具有相关联的数据，该数据是由返回的部分选定继承，并自动绑定到新选定的元素。如果多个元素匹配选择器，只在文件遍历顺序的第一个匹配的元素将被选中。 This method is approximately equivalent to: 这个方法大约相当于： 其中，selection 是当前过渡的隐含选择器。另外，返回的新过渡从当前的过渡继承缓动，持续和延迟。

transition.**selectAll**(*selector*)

对当前过渡的每个元素，选择匹配指定选择字符串后代元素。返回的选择按当前选择的祖先节点进行分组。如果当前元素没有元素匹配指定选择器，返回的选择中当前索引的组将是空的。部分选取不从当前选择继承数据。然而，如果数据之前绑定到选定的元素上，那个数据可用于操作符。 This method is approximately equivalent to: 这个方法大约相当于：

```
selection.selectAll(selector).transition()
```

其中，selection 是当前过渡的隐含选择器。另外，返回的新过渡从当前的过渡继承缓动，持续和延迟。当前过渡中每个子元素的持续和延迟都是继承自父元素。

transition.**filter**(*selector*)

过滤过渡，返回一个新的过渡只包含其指定的选择是正确的元素。选择器可以被指定为一个函数或选择字符串，如“foo”。和其他操作符一样，该函数被传递当前数据d和当前索引i，this上下文作为当前的DOM元素。类似内置的数组过滤函数，返回选择不保留原选择的索引；返回移除元素的副本。如果您想保留索引，使用select代替。例如，选择所有其他元素：

```
var odds = transition.select(function(d, i) { return i & 1 ? this : null; });
```

Equivalently, using a filter function: 等价于使用过滤函数：

```
var odds = transition.filter(function(d, i) { return i & 1; });
```

Or a filter selector: 或者一个过滤选择器：

```
var odds = transition.filter(":nth-child(odd)");
```

Thus, you can use either select or filter to apply tweens to a subset of elements. 这样你就可以使用select和过滤器在元素子集上应用补间了。

transition.**transition**()

在同样的选中元素上创建一个新的选择，在这个过渡结束后启动。新的过渡继承当前过渡的持续和缓动。这可以用来定义链式过渡，而无需监听“end”事件。

// **revision needed** 需要重新翻译

```
Creates a new transition on the same selected elements that starts when this transition e

As described above, the delay state of chained transitions is used as a scaffold for the


selection.transition() // this transition runs from t=1s to t=3s
    .delay(1000)
    .duration(2000)
  .transition() // then a delay from t=3s to t=4s
    .duration(1000)
  .transition() // then lastly another transition from t=4s to t=5s


Though, note that the last transition in this chain inherits the 1s duration from the int
```

Control

transition.**each**([*type*,]*listener*)

如果指定了类型（*type*），就为过渡事件增加一个侦听器(listener)，同时支持“start”, “end”和"interrupt"件。侦听器会被过渡中每个单独的元素调用。

*start*事件在过渡的第一个异步回调（tick）中被调用，于任何补间(tween)被调用之前。对于0延迟过渡，通常在过渡之后的17ms被调用。状态函数对触发每个元素的瞬间变化是很有用的，例如改变不能被打断的属性。

*end*事件在过渡的持续和延迟结束之后最后一个异步回调（tick）中被调用，在所有的补间都使用t=1调用之后。注意，如果对于一个给定的元素过渡被之后调度的过渡取代了， end事件就不会分配到这个元素上；中断过渡不会触发结束事件。例如，[[transition.remove|#remove]]

调度的每个元素在过渡结束时被除去，但如果过渡被中断了，元素将不被删除。end事件可以被用作替代[[transition.transition|#transition]],通过选择当前元素this，并导出新过渡来创建过渡链(chained transitions)。

translation needed: 需要翻译:

```
The *interrupt* event is invoked if an active transition is interrupted by another transi
```

如果没有指定type，行为类似于[[selection.each|Selections#each]]：直接为当前过渡的每个元素调用指定的函数，通过在当前的数据d和索引i，与当前的DOM元素的this上下文。transition.each将从父过渡继承过渡参数，包括id，延迟和缓动。因此，transition.each内创建过渡不会打断父过渡，类似subtransitions。

transition.each方法可用于链接过渡并在一组过渡上分享定时（timing）。例如：

```
d3.transition()
    .duration(750)
    .ease("linear")
    .each(function() {
        d3.selectAll(".foo").transition()
            .style("opacity", 0)
            .remove();
    })
    .transition()
    .each(function() {
        d3.selectAll(".bar").transition()
            .style("opacity", 0)
            .remove();
    });
```

通过使用transition.each中的 `d3.select(this)` ，你甚至可以继承一组选定元素的交错延迟。这个技术在数轴组件中用来支持自动过渡。此方法被用于[[SVG 轴|SVG-轴]]来支持自动过渡(automatic transitions).

transition.**call**(*function*[, *arguments*…])

调用指定的函数（*function*）一次，通过一些可选的参数（*arguments*）传递当前的过渡。call操作符总是返回当前的过渡，与指定函数的返回值无关。call操作符和手动执行一个函数是一样的。但是，它更容易使用方法链。例如，假设我们要在许多不同的地方以同样的方式设置一些属性。我们编写代码，把它封装为一个可重复使用的函数：

```
function foo(transition) {
    transition
        .attr("name1", "value1")
        .attr("name2", "value2");
}
```

现在我们可以这样使用 `foo()` ：

```
foo(d3.selectAll("div").transition());
```

或者, 等价地：

```
d3.selectAll("div").transition().call(foo);
```

在很多情况下，是可以在过渡和选择上调用同一个函数foo的，因为在过渡和选择上是一样的方法。被调函数的 `this` 就是当前过渡。第一个用法略显多余，这我们可能在未来解决。

transition.**empty**()

返回true如果当前过渡是空的；过渡是空的，是指它不包含任何非null元素。

transition.**node**()

返回在当前过渡的第一个非空元素。如果转换为空，则返回null。

transition.**size**()

返回在当前的过渡元素的总数。

Easing

d3.**ease**(*type*[, *arguments*…])

返回一个指定类型（type），带有任何可选参数（arguments）的内置缓动函数。一个缓动函数将当前参数化的时间值t从定义域[0,1]映射到一个相似返回的其他值；这通常用来设置过渡的缓动。D3支持以下的缓动类型：• linear -标识函数, t. • poly(k) – t 的 k次方 (例如3). • quad – 等价于poly(2). • cubic - 等价于poly(3). • sin – 使用三角函数 sin. • exp – 2的t次方 • circle - 四分之一圈 • elastic(a, p) - 模拟一个橡皮筋；略有延长可能会超出0,1。 • back(s) - 模拟备份到一个停车位。 • bounce - 模拟一个有弹性碰撞。 这些内置的类型可以采用各种方式进行扩展：• in - 标识的功能。 • out -逆转缓动的方向为[1,0]。 • in-out -从[0,.5]和[.5,1]复制和镜像缓动函数。 • out-in -从[1,.5]和[.5,0]复制和镜像缓动函数。 默认的缓动函数是“cubic-in-out” 它提供了适合慢入慢出动画。

ease(*t*)

给定的参数时间t，通常在范围[0,1]内，返回的缓动的时间。返回的值通常是在范围[0,1]为好，但对于某些缓动的函数也可以超出这个范围，比如“弹性（elastic）”就可稍微延伸。

Timers

D3在内部维护一个高效的定时器队列，使成千上万的定时器可以用最小的开销并发地处理；此外，该定时器队列保证动画的一致的定时时同时或分阶段转换被调度。如果您的浏览器支持它，定时器队列将使用requestAnimationFrame流体高效的动画。定时器队列也是聪明的使用setTimeout的时候有一个调度事件之前，出现长时间的延迟。

d3.**timer**(*function*[, *delay*[, *time*]])

启动一个自定义动画计时器，重复地调用指定的函数（function ），直到它返回true。计时器启动后没有办法把它取消，所以一定要确保完成时，你的计时器函数返回true！ 当给定函数将在一段延迟之后被调用时，一个以毫秒为单位的可选数字delay可能被指定。延时是相对于指定时间从UNIX纪元以毫秒为单位;如果没有指定时间，则默认为Date.now。 您可以使用延迟（delay ）和时间（time ）以指定function应该开始被调用的相对和绝对时刻。例如，一个日历通知可能被编码为： d3.timer(notify, -4 *1000* 60 * 60, +new Date(2012, 09, 29)); // four hours before midnight October 29 (months are zero-based)

d3.timer.**flush**()

立即执行（调用一次）任何活动的计时器。通常，在瞬时延迟（<10毫秒）之后执行零延迟过渡。如果浏览器两次呈现页面，这可能会导致一个简短的闪烁。一旦在第一事件循环的结束，然后再次紧接在第一定时器的回调。通过在第一事件循环结束时刷新定时器队列，可以立即执行任何零延迟过渡，避免了闪烁。

Interpolation

D3有很多内置interpolators来简化任意值的过渡；插值器是一个函数，用来将值域[0,1]中参数值t映射为一种颜色，数字或任意值。

d3.**interpolate**(*a*, *b*)

返回一个介于a和b之间的默认插值器。插值器的类型是基于后面一个值b的类型,使用以下算法: 1.如果b是颜色（color）类型，则返回interpolateRgb插值器。 2.如果b是字符串（string）类型，则返回interpolateString插值器。 3.如果b是数组（array）类型，则返回interpolateArray插值器。 4.如果b是对象（object）类型，且不能强制转换为数字类型, 则返回interpolateObject插值器。 5.否则，返回interpolateNumber插值器。 基于选定的插值器,a将被强制转换为一个适当的对应类型。颜色检查适用于 d3.rgb和其他颜色空间以及/^(#|rgb(|hsl()/形式的颜色字符串或CSS指定的颜色。 这个默认插值器的行为可以扩展至支持其他的类型。只要添加用户自定义插值器到d3.interpolators数组中即可。

interpolate(*t*)

对在区间[0,1]中一个给定的参数t，返回相关的插入值。插值器通常用结合比例尺一起使用，映射一个输入域(如定量维度)到一个输出范围(如一系列颜色或像素位置)。

d3.**interpolateNumber**(*a*, *b*)

返回一个a，b两个数字之间的数字插值器。返回的插值器相当于:

```
function interpolate(t) {
  return a * (1 - t) + b * t;
}
```

注意:当插值器用于生成一个字符串时(例如attr)，应该避免插入0或从0返回插值器。当使用字符串转化时，非常小的值可能转化为科学记数法格式从而产生一个暂时无效的属性或样式属性值。例如,数字0.0000001便转换为字符串"1 e-7"。 当插入不透明度值时，这一点尤其明显。为了避免科学记数法导致过渡的开始和结束值都是1e-6这种现象，此值是不用指数表示法转化为字符串的最小的。

d3.**interpolateRound**(*a*, *b*)

返回一个a和b两个数字之间的数字插值器；插值器类似于interpolateNumber,除了它会将结果值四舍五入为最近的整数。

d3.**interpolateString**(*a*, *b*)

返回一个a和b两个字符串之间的字符串插值器。字符串插值器寻找数字嵌入在a和b里,其中每个数字的形式如下:

```
/[-+]?(?:\d+\.?\d*|\.?\d+)(?:[eE][-+]?\d+)?/g
```

对于嵌入到b的每个数字,插值器将尝试从a中找到一个相应的数字。如果找到相应的数,便通过使用interpolateNumber创建一个数字插值器。字符串b的其余部分将作为一个模板：b字符串的静态部分对于插值器将保持值不变，插入的数字值将嵌入到模板。例如,如果a是"300 12 px sans-seri"，b是"500 36 px Comic-Sans"，两个嵌入的数字被发现。字符串剩下的静态部分是两个数字之间的空间(" "),和后缀("px Comic-Sans")。当传入值t =0.5时， 插值器的结果是"400 24px Comic-Sans"。

d3.**interpolateRgb**(*a*, *b*)

返回一个a和b两种颜色值之间的RGB颜色空间插值器。颜色a和b不需要在RGB里，但他们将通过d3.rgb转换为RGB值。红、绿、蓝通道是线性地插入值，在某种程度上相当于interpolateRound，即小数部分的通道值是不允许返回的。插值器的返回值是一个十六进制RGB字符串。

d3.**interpolateHsl**(*a*, *b*)

返回一个a和b两种颜色之间的HSL颜色空间插值器。颜色之间的a和b不需要在HSL范围中，但他们会通过d3.hsl被转换成HSL值。色相、饱和度和明度是线性的来插入值，在某种程度上相当于interpolateNumber。(使用开始和结束色调之间的最短路径)。插值器的返回值是一个十六进制RGB字符串。

d3.**interpolateLab**(*a*, *b*)

返回一个a和b两种颜色之间的Lab颜色空间插值器。颜色a和b将会通过d3.lab（如果需要的话）被转换成Lab值。然后颜色通道是线性地插入值，在某种程度上相当于interpolateNumber。插入器的返回值是一个十六进制RGB字符串。

d3.**interpolateHcl**(*a*, *b*)

返回一个a和b颜色之间的HCL颜色空间插值器。颜色a和b将会通过d3.hcl（如果需要的话）被转换成HCL值。然后颜色通道线性地插入值，在某种程度上相当于interpolateNumber。(使用开始和结束色调之间的最短路径)。插值器的返回值是一个十六进制RGB字符串。 # d3.**interpolateArray**(*a*, *b*)

返回一个在两个数组a和b之间的数组插值器。一个拥有与b相同长度的数组模板将被创建。对于b中的每一个元素，如果在a中存在一个相应的元素，那么对于这两个元素就会有一个通用的插值器将通过interpolate创建。如果没有这样的元素,那么来源于b的静态值将在这个模板中被使用。然后，对于给定的参数t，模板的嵌入的插值器将被求值。然后返回更新后的数组模板。例如,如果a数组为[0,1]，b数组为[1,10,100]，然后当参数t =0.5时，插值器的结果便是数组[0.5,5.5,100]。 注意:创建一个模板数组的非保护性拷贝；返回数组的修改将对内插器的随后的评价产生不利影响。不复制是为了保证插值器的快速，因为它们是动画内部循环的一部分。

d3.**interpolateObject**(*a*, *b*)

返回一个于a和b两个对象之间的插值器。一个拥有与b相同属性的对象模板将被创建。对于b中的每个属性，如果在a中存在一个对应的属性，那么通过interpolate创建这两个元素通用的插值器。如果没有这样的属性，来源于对象b的静态值将被使用在模板里。然后，对于给定的参数t，模板的嵌入的插值器将被进行求值。然后返回更新后的数组模板。例如，如果a对象是{ x:0,y:1 }和b对象是{ x:1,y:10 z:100 },那么当参数t =0.5时插值器的结果便是对象{ x:5,y:5.5,z:100 }。 对于数据空间差值，当插入数据值而不是属性值时，对象插值器尤其有用。例如，您可以插入一个对象，用来描述一个饼图里的弧，然后使用d3.svg.arc来计算这个新的SVG路径数据。 注意:创建一个模板数组的非保护性拷贝；返回数组的修改将对内插器的随后的评价产生不利影响。不复制是为了保证插值器的快速，因为它们是动画内部循环的一部分。

d3.**interpolateTransform**(*a*, *b*)

返回一个由两个a和b表示的二维仿射变换的插值器。每个变换分解成一个标准的转换，旋转，x偏移和比例；然后插入这些转换分量。这种行为是由CSS规范的：详看matrix decomposition for animation。

d3.**interpolateZoom**(*a*, *b*)

基于Jarke J. van Wijk and Wim A.A. Nuij共同开发的“Smooth and efficient zooming and panning”（平滑有效的缩放和移动），返回一个于两个二维平面视图a和b之间的平滑插值器。每个视图的定义是由三个数字构成的数组：cx，cy和width。前两个坐标cx，cy代表视窗

的中心；最后width（宽度）代表视窗的大小。返回的插入器还有一个持续时间（duration ）属性，此属性推荐用以毫秒为单位的过渡时间来编码。这个持续时间是基于x，y空间弧形轨迹路径长度的。如果你想更慢或更快的转换，那么就通过一个任意的比例因子与此相乘(*V*在原始论文中有描述)。

d3.geo.**interpolate**(*a*, *b*)

详见d3.geo.interpolate。

d3.**interpolators**

内置插值器工厂的数组，是d3.interpolate所使用的。额外的插值器工厂可能被添加到这个数组的末尾端。如果它支持可以插入两个指定的输入值，那么每个工厂可能会返回一个插值器；否则，工厂将返回一个假值并尝试返回其他的插值器。 例如，注册一个自定义插值器用来格式化美元和美分，你可能会写:

```
d3.interpolators.push(function(a, b) {
    var re = /^\$([0-9,.]+)$/, ma, mb, f = d3.format(",.02f");
    if ((ma = re.exec(a)) && (mb = re.exec(b))) {
        a = parseFloat(ma[1]);
        b = parseFloat(mb[1]) - a;
    return function(t) {
    return "$" + f(a + b * t);
        };
      }
});
```

然后， `d3.interpolate("$20", "$10")(1/3)` ，返回 $16.67 ；`d3.interpolate("$20", "$10")(1)` ，返回 $10.00 ； `d3.interpolate("$20", "$10")(0)` ，返回 $20.00。
Interpolation部分翻译来自Harry翻译的校正。 1-16页 咕噜翻译 2014-11-15 21:36:38 17-26页 Harry译 2014-03-30

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 数组

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

当你使用D3处理数据可视化时，通常会倾向于做大量的数组操作**(array manipulation)**。那是因为数组是D3的标准的数据呈现形式。数组处理的一些常见形式包括：取数组的一个连续片段（子集），使用判定函数过滤数组，使用变换函数映射数组为一组平行的值。在看到D3框架提供的一系列方法处理数组时，你应当很熟悉强大的JavaScript内置的数组的方法.

JavaScript包含修改数组的赋值方法**(mutator methods)**：

- array.pop -删除数组最后一位元素。
- array.push - 往数组的末尾新增一个或多个元素。
- array.reverse - 把数组元素的逆转顺序。
- array.shift - 删除数组第一位元素。
- array.sort - 给数组排序。
- array.splice - 给数组添加或者删除元素。
- array.unshift - 往数组的第一位新增一个或多个元素。

还有一些数组的存取方法**(accessor methods)**，返回数组的一些描述:

- array.concat - 合并数组或合并数组的值。
- array.join - 合并数组所有元素拼接成字符串。
- array.slice - 提取数组的一个选择。
- array.indexOf - 定位到数组第一个值。
- array.lastIndexOf - 定位到数组内最后一个值。

最后，对数组中的元素使用用函数的迭代方法**(iteration methods)**:

- array.filter - 由满足特定条件的元素创建一个新的数组。
- array.forEach - 为数组中每一个元素调用一个函数。
- array.every - See if every element in the array satisfies a predicate.
- array.map - Create a new array with the result of calling a function on every element in the array.
- array.some - See if at least one element in the array satisfies a predicate.
- array.reduce - Apply a function to reduce the array to a single value (from left-to-right).
- array.reduceRight - Apply a function to reduce the array to a single value (from right-to-left).

排序 （Ordering）

d3.**ascending**(*a*, *b*)

如果 `a < b` 返回-1， `a > b` 返回1， `a = b` 返回0。 这是固有的比较器方法，也可用于关联内置数组排序的方法来给元素升序排序：

```
function ascending(a, b) {
  return a < b ? -1 : a > b ? 1 : a >= b ? 0 : NaN;
}
```

注意，如果没有给数组的内置排序方法没有指定比较器函数，那默认的排序是字典排序(按字母顺序排序)，而非自然排列！所以当以数组的数字来排序时会导致bug。

d3.**descending**(*a*, *b*)

如果 `a > b` 返回-1， `a < b` 返回1， `a = b` 返回0。 这是固有的比较器方法，也可用于关联内置数组排序的方法来给元素降序排序：

```
function descending(a, b) {
  return b < a ? -1 : b > a ? 1 : b >= a ? 0 : NaN;
}
```

注意，如果没有给数组的内置排序方法没有指定比较器函数，那默认的排序是字典排序(按字母顺序排序)，而非自然排列！所以当以数组的数字来排序时会导致bug。

d3.**min**(*array*[, *accessor*])

返回给定数组（array）中自然排序最小的值。如果数组为空，返回undefined。如果指定了accessor 参数，等同与在计算最小值之前调用了array.map(accessor)方法。不同于内置的Math.min，这个方法会忽略未定义的值；这对比例尺([[d3.scale|比例尺]])定义域计算很有用处，当只考虑数据的定义区域。另外，元素的比较用的是自然排序而不是数字排序。例如，["20","3"]的最小值是20，然而[20,3]的最小值是3。

d3.**max**(*array*[, *accessor*])

返回给定数组（array）中自然排序最大的值。如果数组为空，返回undefined。如果指定了accessor 参数，等同与在计算最大值之前调用了array.map(accessor)方法。而并非内置的Math.max，这个方法会忽略未定义的值；这对当只需要定义数据的区域的比例尺定义域计算很有用处。另外，元素的比较用的是自然排序而不是数字排序。例如，["20","3"]的最大值是3，然而[20,3]的最大值是20。

d3.**extent**(*array*[, *accessor*])

返回给定数组（array）自然排序的最小值和最大值，等同于同时调用d3.min和d3.max.

d3.**sum**(*array*[, *accessor*])

返回给定数组（array）的和。如果数组为空，返回 0。可选参数accessor函数 被指定，等同于在计算和之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)；当只考虑明确定义的值时，这个方法可用于计算数据的和。

d3.**mean**(*array*[, *accessor*])

返回给定数组（array）的平均数。如果数组为空，返回 `undefined` 。可选参数accessor函数被指定，等同在计算平均数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)，当只考虑明确定义的值时这个方法计算数据和是很有用的。

d3.**median**(*array*[, *accessor*])

返回给定数组（array）以R-7算法得出的中位数。如果数组为空，返回 undefined。可选参数 accessor 被指定，等同在计算中位数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)，当只考虑明确定义的值时这个方法计算数据和是很有用的。

d3.**quantile**(*numbers*, *p*)

返回给定数组*numbers*的*p*分位数，其中p 是一个0到1范围的数。例如，中位数可以由p = 0.5计算，第一个四分位数是p = 0.25，第三个四分位数是p = 0.75。这个特别实现了R-7算法，这是R编程语言和Excel默认的方式。这个方法需要数组numbers包含数字且数字升序顺序排列，例如使用 d3.ascending排序。

```
var a = [0, 1, 3];
d3.quantile(a, 0); // return 0
d3.quantile(a, 0.5); // return 1
d3.quantile(a, 1); // return 3
d3.quantile(a, 0.25); // return 0.5
d3.quantile(a, 0.75); // return 2
d3.quantile(a, 0.1); // return 0.19999999999999996
```

d3.**variance**(*array*[, *accessor*])

返回给定数组（array）的无偏总体方差（unbiased estimator of the population variance）。如果数组的长度小于2，返回 `undefined` 。可选参数accessor 被指定，等同在计算中位数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)。

d3.**deviation**(*array*[, *accessor*])

返回给定数组（array）的标准差，即方差（bias-corrected variance）的平方根。如果数组的长度小于2，返回 `undefined` 。可选参数accessor 被指定，等同在计算中位数之前调用*array.map(accessor)* 。此方法忽略无效值(如 `NaN` 和 `undefined`)。

d3.**bisectLeft**(*array*, *x*[, *lo*[, *hi*]])

定位数组 *array* 中的 *x* 的插入点，以保持已有序列。参数 *lo* 和 *hi* 用来指定数组的子集；默认情况下整个数组都被使用。如果 *x* 在 *array* 中已存在，插入点在所有元素之前（左侧）。返回值适合用作拼接（splice）已经排序的数组*array* 的第一个参数。返回的插入点i把*array* 分为两个区：数组中所有*array*.slice(lo, i)中 `v < x` 的v在左边，数组中所有 *array*.slice(i, hi)中 `v >= x` 的v在右边。

d3.**bisect**(*array*, *x*[, *lo*[, *hi*]])</br> # d3.**bisectRight**(*array*, *x*[, *lo*[, *hi*]])

和bisectLeft类似，但返回插入点来自于数组*array*中任意实体*x*之后（右侧）。返回的插入点*i*把*array* 分为两个区：数组中所有*array*.slice(lo, i)中 `v <= x` 的v在左边，数组中所有*array*.slice(i, hi)中 `v > x` 的v在右边。

d3.**bisector**(*accessor*)
d3.**bisector**(*comparator*)

使用指定参数*accessor*或者*comparator* 函数返回一个二等分线。返回的对象有 `left` 和 `right` 属性，分别类似于bisectLeft和bisectRight方法。这个方法能用于二等分对象数组而不适用于原始的简单数组。例如下列对象的数组:

```
var data = [
  {date: new Date(2011,  1, 1), value: 0.5},
  {date: new Date(2011,  2, 1), value: 0.6},
  {date: new Date(2011,  3, 1), value: 0.7},
  {date: new Date(2011,  4, 1), value: 0.8}
];
```

一个合适的二等分函数可定义为：

```
var bisect = d3.bisector(function(d) { return d.date; }).right;
```

然后调用 `bisect(data, new Date(2011, 1, 2))` ，返回索引。 如果你想使用不同于自然排序的方法对值进行排序，那么可以使用比较器（comparator）而不是访问器（accessor），例如降序排序而不是升序排序的时候。

```
var bisect = d3.bisector(function(a, b) { return a.date - b.date; }).right;
```

d3.**shuffle**(*array*[, *lo*[, *hi*]])

使用Fisher–Yates shuffle来把传入参数*array*随机排序.

关联数组 （Associative Arrays）

关联数组(字典)和数组类似，由以名称作为键的字段和方法组成。 它包含标量数据，可用索引值来单独选择这些数据，和数组不同的是， 关联数组的索引值不是非负的整数而是任意的标量。这些标量称为Keys， 可以在以后用于检索数组中的数值. JavaScript 中另一种常见数据类型就是关联数组，或者简单说就是具有一系列命名属性的对象。在Java中简称映射(键值对) map，而在Python中称为字典dictionary。JavaScript为关联数组中键（属性名称）的迭代提供一个标准机制：那就是 for…in loop。然而，注意迭代的次序是未定义的。D3提供了一些将关联数组转化为索引数组的方法。

d3.**keys**(*object*)

返回一个包含指定对象(关联数组) 属性名称的数组。返回数组的顺序未定义。

d3.**values**(*object*)

返回一个包含指定对象(关联数组) 属性值的数组。返回数组的顺序未定义。

d3.**entries**(*object*)

返回一个包含对象(*object*)(一个关联数组)中名称以及值(键和值， key and value)的数组(array)。每一个实体都是有键值对的对象，例如 `{key: "foo", value: 42}` 。返回数组的顺序未定义。

```
d3.entries({foo: 42,  bar: true}); // returns [{key: "foo", value: 42}]
```

映射（**Maps**）

当你尝试在JavaScript中用空对象作为map，当内部属性名称(键)作键时，会导致意外的行为(unexpected behavior)。比如，当你设置 `object["__proto__"] = 42` 时，最终不会达到你理想中的结果。又如你尝试查询给定key是否在map中定义了；"hasOwnProperty" in object 返回true，因为空对象(从对象原型)继承了hasOwnProperty方法。为避免这些问题，ES6提出了简单映射和集合(simple maps and sets) 理论。直到现代浏览器支持了这些集合，你可以使用[d3.map]替代.

注意：不同于建议的 ES6 map， d3.map的key仍然强制使用字符串，而不是严格的相等。

d3.**map**([*object*][, *key*])

构建一个新的map，如果指定参数*object*，复制参数*object*对象内所有枚举属性到map中。参数对象可能是数组. 可以使用一个键*key*函数来计算数组里每个数值的键. 如下:

```
var m = d3.map([{name: "foo"}, {name: "bar"}], function(d) { return d.name; });
m.get("foo"); // {"name": "foo"}
m.get("bar"); // {"name": "bar"}
m.get("baz"); // undefined
```

参见 d3.nest。

map.**has**(*key*)

当且仅当map有指定key的实体时返回true。注意：该值可能是 `null` 或 `undefined` 。

map.**get**(*key*)

返回参数*key*的值。如果map中没有参数*key*相同元素，返回 `undefined` 。

map.**set**(*key*, *value*)

指定*key*的*value* ；返回新的*value*。如果map之前同样的*key* 有一个实体了，那么旧实体被新值替代。

map.**remove**(*key*)

若map有指定*key*的实体，删除此实体并返回 `true` 。否则，此方法不做任何操作，返回 `false` 。

map.**keys**()

返回在这个map所有的键的数组。返回键的集合顺序是随机的。

map.**values**()

返回在这个map所有的值的数组，返回值的集合顺序是随机的。

map.**entries**()

返回一个map内所有键-值对象的数组。返回元素的集合顺序是随机的。任何元素的键必须是字符串类型，但值可为任何类型。

map.**forEach**(*function*)

给map中每个元素调用一个指定函数*function*，传递元素的键和值作为两个参数。*function*的使用的 `this` 指针将指向这个map。返回 `undefined` 。迭代的顺序是随机的。

map.**empty**()

返回 `true` 当且仅当map中没有元素。

map.**size**()

返回map中元素的个数

集合（Sets）

d3.**set**([*array*])

新建一个集合，如果指定了*array*，添加*array* 的字符串值到返回集合中。

set.**has**(*value*)

当且仅当集合中具有指定参数*value* 字符串相同的实体，返回 `true` 。

set.**add**(*value*)

添加指定参数*value* 字符串到集合中.

set.**remove**(*value*)

如果集合中含有指定参数*value* 字符串相同元素，返回 `true` 并删除元素。否则，这个方法不做任何操作，并返回 `false` 。

set.**values**()

返回一个由集合中所有字符串类型值组成的数组。数组中的值的顺序为随机的。可作为集合中唯一值的简便计算方法(去重)。例如:

```
d3.set(["foo", "bar", "foo", "baz"]).values(); // "foo", "bar", "baz"
```

set.**forEach**(*function*)

给集合中每个元素调用一个指定*function*，传递元素的值作为参数。*function*的使用的 `this` 指针将指向这个map。返回 `undefined` ，迭代的顺序是随机的。

set.**empty**()

当且仅当集合中没有值，返回 `true` 。

set.**size**()

返回集合中值的个数。

数组运算符 (Array Operators)

d3.**merge**(*arrays*)

合并指定参数*arrays* 为一个数组，此方法和数组内置方法 `concat` 类似；唯一不同是当你要处理二维数组时， `d3.merge(arrays)` 方法更方便。

```
d3.merge([ [1], [2, 3] ]); // returns [1, 2, 3]
```

d3.**range**([*start*,]*stop*[, *step*])

生成一个包含算数级数的数组，类似于Python的内置函数range。这个方法常用来遍历一个数字序列或者整型数值。例如数组中的索引。不同于Python版本，这个参数不必是整形。尽管如果它们是浮点精度类型时这个结果更加可预测。如果省略*step*，默认值是1。如果省略*start*参数，默认值就是0。结果中不包含*stop*值。完整的形式是返回一个数字数组 `[start,start+step,start+2 *step,…]` 。如果*step*是正的，则最后一个元素是小于 `stop` 的 `start+ i*step` 中的最大数值；如果*step*是负的，最后一个元素是大于 `stop` 的 `start + i*step` 中的最小数值。如果返回的数组将包含值无限多数字，就会抛出一个错误，而不是造成无限循环。

d3.**permute**(*array*, *indexes*)

使用指定的*indexes*数组返回指定数组的转置。返回数组包含indexes数组中索引对应的元素，按顺序。例如， `permute(["a", "b", "c"], [1, 2, 0])` 返回 `["b", "c", "a"]` 。indexes数组的长度和array中的元素长度不一样是可以接受的，并且允许indexes数组重复或者省略。 这

个方法可以用来按固定顺序提取对象中的值到一个数组中。（在JavaScript中indexes数组是和 `.length` 有特殊关系的简单属性）。按顺序提取带键的值可以用来生成嵌套选择中的数据数组。例如，我们可以用表格形式展示上述的一些明尼苏达州大麦产量数据：

```
var cols = ["site", "variety", "yield"];
thead.selectAll('th').data(cols)
    .enter().append('th').text(function (d) { return d.toUpperCase(); });
tbody.selectAll('tr').data(yields)
    .enter().append('tr').selectAll('td').data(function (row) { return d3.permute(row, co
        .enter().append('td').text(function (d) { return d; });
```

d3.**zip**(*arrays…*)

返回的数组的数组，其中，第i个数组包含来自每个*arrays*参数的第i个元素。返回的数组长度被截断为*arrays*的最短的数组的长度。如果*arrays*只包含一个数组，则返回的数组是包含一个元素的数组。不带任何参数，则返回的数组是空的。

```
d3.zip([1, 2], [3, 4]); // returns [[1, 3], [2, 4]]
```

d3.**transpose**(*matrix*)

等价于 `d3.zip.apply(null, matrix)` ；使用zip操作符作为二维矩阵变换(matrix transpose)。

d3.**pairs**(*array*)

对指定参数array中元素的每个相邻对，返回元组(元素i和元素i-1)的新数组。例如：

```
d3.pairs([1, 2, 3, 4]); // returns [[1, 2], [2, 3], [3, 4]]
```

如果指定参数array 中少于两个元素，则返回一个空数组。

嵌套 (Nest)

嵌套允许数组中的元素被组织为分层树型结构；类似SQL语句里面的GROUP BY方法，但不能多级别分组，而且输出的结果是树而不是一般的表。树的层级由key方法指定。树的叶节点可以按值来排序，而内部节点可以按键来排序。可选参数汇总（rollup）函数可以使用加法函数瓦解每个叶节点的元素. nest 操作符(d3.nest返回的对象)是可以重复使用的，不保留任何嵌套数据的引用。

例如，思考下面1931-2年间明尼苏达州（美国州名）麦田地皮的表格数据结构：

```
var yields = [{yield: 27.00, variety: "Manchuria", year: 1931, site: "University Farm"},
              {yield: 48.87, variety: "Manchuria", year: 1931, site: "Waseca"},
              {yield: 27.43, variety: "Manchuria", year: 1931, site: "Morris"}, ...]
```

为了方便查看，可以嵌套元素首先按year然后按variety，如下：

```
var nest = d3.nest()
    .key(function(d) { return d.year; })
    .key(function(d) { return d.variety; })
    .entries(yields);
```

返回的嵌套数组中。每个元素的外部数组是键-值对，列出每个不同键的值:

```
[{key: 1931, values: [
    {key: "Manchuria", values: [
        {yield: 27.00, variety: "Manchuria", year: 1931, site: "University Farm"},
        {yield: 48.87, variety: "Manchuria", year: 1931, site: "Waseca"},
        {yield: 27.43, variety: "Manchuria", year: 1931, site: "Morris"}, ...]},
    {key: "Glabron", values: [
        {yield: 43.07, variety: "Glabron", year: 1931, site: "University Farm"},
        {yield: 55.20, variety: "Glabron", year: 1931, site: "Waseca"}, ...]}, ...]},
 {key: 1932, values: ...}]
```

嵌套的形式可以在SVG或HTML很容易迭代和生成层次结构。

有关 `d3.nest` 详见:

- Phoebe Bright's D3 Nest Tutorial and examples
- Shan Carter's Mister Nester

d3.**nest**()

创建一个新操作符。keys集合初始为空。如果map或entries 操作符在任何键函数被注册之前被调用，这个嵌套操作符通常返回输入数组。例如 http://bl.ocks.org/phoebebright/raw/3176159/

nest.**key**(*function*)

注册一个新的键函数*function*。键函数将被输入数组中的每个元素调用，并且必须返回一个用于分配元素到它的组内的字符串标识符。通常，这个函数被实现为一个简单的访问器，如上面例子中year和variety的访问器。 输入的数组的引导（index）并没有传递给*function*。每当一个key 被注册，它被放到一个内部键数组的末端，和由此产生的map或实体将有一个额外的层级。当前没有一个方法可以删除或查询注册的键。最近注册的键在后续的方法中被当作当前键。

nest.**sortKeys**(*comparator*)

使用指定的参数*comparator*来给当前键值排序，等效于d3.descending。如果没有指定comparator 参数键的排序则返回undefined。注意：只影响实体操作符的结果；map操作符返回键的顺序永远是undefined，无论什么比较器。

```
var nest = d3.nest()
    .key(function(d) { return d.year; })
    .sortKeys(d3.ascending)
    .entries(yields);
```

nest.**sortValues**(*comparator*)

使用指定的*comparator*参数给叶子元素排序，等效于d3.descending。这相当于在应用nest操作符前给输入的数组排序；然而，当每一组更小时它通常是更有效的。如果没有指定值的比较器，元素则按输入数组中显示的顺序排列。通常用于map和实体（entries）的操作符。

nest.**rollup**(*function*)

为每组中的叶子元素指定汇总函数（rollup ）*function*。汇总函数的返回值会覆盖叶子值数组。不论是由map操作符返回的关联数组，还是实体操作符返回的每个实体的值属性。

nest.**map**(*array*[, *mapType*])

对指定的数组使用nest操作符，返回一个关联数组。返回的关联数组*array*中每个实体对应由第一个key函数返回的不同的key值。实体值决定于注册的key函数的数量：如果有一个额外的key，值就是另外一个嵌套的关联数组；否则，值就是过滤自含有指定key值得输入数组*array*的元素数组。

如果指定了*mapType*，指定的函数就会用来构造一个map而不是返回一个简单的JavaScript对象。推荐使用d3.map 实现这个目的，例如：

```
var yieldsByYearAndVariety = d3.nest()
    .key(function(d) { return d.year; })
    .key(function(d) { return d.variety; })
    .map(yields, d3.map);
```

使用d3.map而不是一个对象提供便捷（例如：返回的map含有 keys 和 values 函数），防止不寻常的键名与JavaScript内置的属性冲突，例如 `__proto__` 。

nest.**entries**(*array*)

为指定的*array*参数应用nest操作符，返回一个键值对数组。从概念上讲，这和对 map 返回的关联数组应用 d3.entries 操作符类似，但是这个是用于每一层而不是只有第一层（最外层）。返回数组中的每个实体对应于第一个key函数返回的不同的key值。实体值取决于注册的key函数的数量：如果有一个关联的key，值就是另外一个嵌套实体数组；否则，值就是含有指定key值得输入数组*array*过滤得到的元素数组。嵌套的案例：http://bl.ocks.org/phoebebright/raw/3176159/

[1] accessor function, 亦为getter, callback, 访问器, 常译作'回调（函数）'。术语翻译 (计算机软件/编程) // Howard Liang 注 Nov 27, 2015

- Carry on 、2014-3-29翻译

- 咕噜2014-11-18翻译，并校对之前的翻译。

> Wiki ▸ API Reference ▸ Core ▸ Math
>
> - 如发现翻译不当或有其他问题可以通过以下方式联系译者:
> - 邮箱：zhang_tianxu@sina.com
> - QQ群：D3数据可视化205076374，大数据可视化436442115

Pseudorandom Number Generation 你可以使用内置函数Math.random来生成统一的分布。例如，成介于0和99 （含）的随机整数，可以调Math.floor(Math.random() * 100)。

d3.random.normal([mean, [deviation]])

返回一个符合正态（高斯）分布normal (Gaussian) distribution的随机数. 随机变量的期望值是mean（默认为0.0），标准差是deviation （默认为1.0）。

d3.random.logNormal([mean, [deviation]])

返回一个满足对数分布 log-normal distribution的随机数. 随机变量自然对数的期望值是mean（默认为0.0），标准差是deviation （默认为1.0）。

d3.random.bates(count)

返回一个符合贝茨分布Bates distribution的随机数。count指定自变量的个数。

d3.random.irwinHall(count)

返回一个符合欧文霍尔分布 Irwin–Hall distribution的随机数。count指定自变量的个数。

2D Transforms

d3.transform(string)

依照SVG的变换属性transform attribute的定义，解析给定的2D仿射变换字符串.。分解这个字符串为一个由 平移、旋转、X偏移和缩放组成的标准表示。此行为规范由CSS定义，参见：matrix decomposition for animation。

transform.rotate

返回此变换的旋转角θ，以度为单位。

transform.translate

返回此变换的[DX，DY]平移，局部坐标（通常为像素）的两元素数组。

transform.skew

返回此变换的x的偏移φ， 单位为度。.

transform.scale

返回变换的[kx, ky]缩放，一个两元素数组。

transform.toString

返回此转换的字符串表示形式，其形式为 "translate(dx,dy)rotate(θ)skewX(φ)scale(kx,ky)"。

*guluT20140326

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 请求

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

如果你不访问数据那么你就不能可视化它。幸运的是有很多的方法可以把数据放到浏览器中。对于小数据集，你可以硬编码到你的脚本里，或者使用数据属性嵌入到DOM中。对于大数据集，你可以引用外部脚本并定义你的数据为一个全局变量。（JSONP就是一个常见的例子）。最通用的方式是使用XMLHttpRequest, 或说XHR加载数据到浏览器。这允许一步加载数据（当数据加载之后其他的页面就可以展示了），并且比JSONP更安全。D3的xhr模块可以简单的加载和解析数据。

当异步加载数据时，代码取决于加载的数据通常存在于回调函数之内。例如D3网站的案例calendar visualization ：http://mbostock.github.com/d3/ex/calendar.html代码不依赖于数据可以在页面加载时立即运行。你也可以发现保存数据到全局命名空间是很方便的，这样你就可以在初始化渲染之后访问它，例如在过渡期间。你可以使用闭包实现它，或者简单地指定加载数据为全局变量。

```
var data; // a global

d3.json("path/to/file.json", function(error, json) {
  if (error) return console.warn(error);
  data = json;
  visualizeit();
});
```

默认情况下，大多数的浏览器不允许跨域访问。为了支持跨域访问，服务器必须设置头Access-Control-Allow-Origin: *。更多信息参见W3C的建议跨域资源分享。对于IE9，d3.xhr使用非标准的XDomainRequest支持跨域访问。 跨域资源分享：http://www.w3.org/TR/cors/

XHR

d3.xhr(url[, mimeType][, callback])

为指定的url创建一个异步访问。参数mimeType可能被指定为第二个参数，例如“text/plain”。如果指定了回调函数，那么请求就会使用GET方法立即发出，并且当资源被加载或者请求失败之后回调函数就会被异步调用。如果有错的话XMLHttpRequest对象代表了response。当错误发生时，response是未定义的。如果response有一个不成功状态值，那么错误就是XMLHttpRequest对象。如果没有指定回调函数，返回的request可以被分发使用xhr.get, xhr.post 或相近的，并使用xhr.on处理。

xhr.header(name[, value])

```
d3.csv("/path/to/file.csv")
    .header("header-name", "header-value")
    .get(function(error, data) {
      // callback
});
```

如果指定了value，设置请求头指定的name参数为指定的value值。如果value参数是null值，就移除指定名称的请求头，如果没指定value值就会返回请求头中指定名称的当前值，请求头名称是大小写敏感的。 请求头只可以在发送请求之前被修改。因此，如果你想指定一个请求头就不能传递回调函数给d3.xhr constructor。而应该使用xhr.get或者相似的。例如：

```
d3.csv("/path/to/file.csv")
    .header("header-name", "header-value")
    .get(function(error, data) {
      // callback
});
```

xhr.mimeType([type])

如果type参数被指定了，就会设置请求的mime类型为指定的值。如果type是null，就会清楚当前的mime类型（有的话）。如果没有指定mime类型，就返回当前的mime类型，默认是null。Mime类型被用来设置"Accept"请求头和覆盖MimeType。请求头只可以在发送请求之前被修改。

xhr.responseType(type)

如果type被指定了，设置response类型，例如：""， "arraybuffer"，"blob"，"document"，或者 "text"。如果type没有指定，就返回当前的response类型默认是""。

xhr.response(value)

如果指定了value参数，就设置response的值函数为指定的函数。如果value没有被指定，返回当前的response的值函数，默认就是验证函数。response的值函数用来映射返回XMLHttpRequest对象为相应的数据类型。例如，对于文本请求，你可以使用function(request) { return request.responseText; }，而对于JSON请求你可能使用function(request) { return JSON.parse(request.responseText); }。

xhr.get([callback])

使用GET方法分发请求。如果指定了回调函数，在发送请求或者出错的时候就会被异步调用。回调函数使用两个参数调用：error（有的话）和response值。错误发生时response值是未定义的。如果没有指定回调函数"load"和"error"监听器会通过xhr.on注册。这个方法是xhr.send的一个方便的包装。

xhr.post([data][, callback])

使用POST方法分发这个请求，在请求体中可选地发送指定的data。如果指定了回调函数，在发送请求或者出错的时候就会被异步调用。回调函数使用两个参数调用：error（有的话）和response值。错误发生时response值是未定义的。如果没有指定回调函数"load"和"error"监听器会通过xhr.on注册。这个方法是xhr.send的一个方便的包装。 使用URL编码的例子：

```
d3.csv("/path/to/file.csv")
    .header("Content-Type", "application/x-www-form-urlencoded")
    .post("a=2&b=3", function(error, data) {
      // callback
    });
```

An example using JSON encoding:

```
d3.csv("/path/to/file.csv")
    .header("Content-Type", "application/json")
    .post(JSON.stringify({a: 2, b: 3}), function(error, data) {
      // callback
    });
```

xhr.send(method[, data][, callback])

使用指定的方法分发这个请求，在请求体中可选地发送指定的data。如果指定了回调函数，在发送请求或者出错的时候就会被异步调用。回调函数使用两个参数调用：error（有的话）和response值。错误发生时response值是未定义的。如果没有指定回调函数"load"和"error"监听器会通过xhr.on注册。这个方法是xhr.send的一个方便的包装。

xhr.abort()

中止正在发送的请求。参见XMLHttpRequest的中止：
http://www.w3.org/TR/XMLHttpRequest/#the-abort%28%29-method

xhr.on(type[, listener])

对指定的类型添加或者移除事件监听器到这个请求。类型必须是以下类型之一： beforesend – 在请求发送之前，允许自定义标题等来进行设定。 progress – 用来监听请求的过程。 load – 当请求成功的完成之后。 error –当请求不成功之后；包含4xx和5xx返回值。

如果一个相同的类型监听器已经被注册，已存在的监听器就会在新监听器添加之前被移除。为了给同一个事件类型注册多个监听器，那么类型将遵循可选的命名空间，例如"load.foo" 和 "load.bar"。为了移除监听器，传递null值为监听器。 如果没有指定监听器，为指定的类型（有的话）返回当前分配的监听器。

简便方法 通常，d3.xhr不会直接使用。取而代之的是使用类型特定的方法，例如： d3.text加载简单文本，d3.json加载JSON，d3.xml 加载XML，d3.html 加载HTML，d3.csv加载逗号分隔值文件，d3.tsv加载制表符分隔文件。

d3.text(url[, mimeType][, callback])

指定的URL创建一个文本文件请求。选项mimeType可以指定为第二参数，例如"text/plain"。如果指定了回调函数，请求将通过GET方法立即分发，当文件被加载或者请求失败之后回调函数将被异步调用。回调函数的调用使用两个参数：error（有的话）和response文本。错误发生时response文本是未定义的。如果没有指定回调函数，返回的请求可能使用xhr.get或近似的方法分发，并使用xhr.on处理。

d3.json(url[, callback])

指定的URL创建一个JSON文件请求。选项mimeType可以指定为第二参数，例如"application/json"。如果指定了回调函数，请求将通过GET方法立即分发，当文件被加载或者请求失败之后回调函数将被异步调用。回调函数的调用使用两个参数：error（有的话）和解析过的JSON。错误发生时解析过的JSON是未定义的。如果没有指定回调函数，返回的请求可能使用xhr.get或近似的方法分发，并使用xhr.on处理。

d3.xml(url[, mimeType][, callback])

指定的URL创建一个XML文件请求。选项mimeType可以指定为第二参数，例如"application/xml"。如果指定了回调函数，请求将通过GET方法立即分发，当文件被加载或者请求失败之后回调函数将被异步调用。回调函数的调用使用两个参数：error（有的话）和解析为文档的XML。错误发生时解析的XML是未定义的。如果没有指定回调函数，返回的请求可能使用xhr.get或近似的方法分发，并使用xhr.on处理。

d3.html(url[, callback])

指定的URL创建一个文本文件请求。选项mimeType可以指定为第二参数，例如"text/html"。如果指定了回调函数，请求将通过GET方法立即分发，当文件被加载或者请求失败之后回调函数将被异步调用。回调函数的调用使用两个参数：error（有的话）和解析为文档碎片的HTML。错误发生时解析后的HTML是未定义的。如果没有指定回调函数，返回的请求可能使用xhr.get或近似的方法分发，并使用xhr.on处理。

d3.csv(url[, accessor][, callback])

指定的URL创建一个CSV文件请求。选项mimeType可以指定为第二参数，例如"text/csv"。如果指定了回调函数，请求将通过GET方法立即分发，当文件被加载或者请求失败之后回调函数将被异步调用。回调函数的调用使用两个参数：error（有的话）和每个RFC 4180解析的行数组。错误发生时行数组是未定义的。如果没有指定回调函数，返回的请求可能使用xhr.get或近似的方法分发，并使用xhr.on处理。

d3.tsv(url[, accessor][, callback])

指定的URL创建一个文本文件请求。选项mimeType可以指定为第二参数，例如"text/tab-separated-values"。如果指定了回调函数，请求将通过GET方法立即分发，当文件被加载或者请求失败之后回调函数将被异步调用。回调函数的调用使用两个参数：error（有的话）和每个RFC 4180解析的行数组。错误发生时行数组是未定义的。如果没有指定回调函数，返回的请求可能使用xhr.get或近似的方法分发，并使用xhr.on处理。

2014年11月22日 00:11:22 gulu翻译

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 格式化

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

格式化数字是不经常用到的，只有在例如丑陋的"0.30000000000000004"出现在你的数轴标签上时，或者你想要使用固定精度将几千的数字组织为更加可读的形式，例如"$1,240.10"，再或者你可能只想展示一个特定的数字的显著位。D3使用标准的数字格式化使得一切变得简单，例如，创建一个用0补齐4位数字的函数，可以这样：

```
var zero = d3.format("04d");
```

现在，你就可以调用 zero 来很方便的格式化你的数字了：

```
zero(2); // "0002"
zero(123); // "0123"
```

当然，除了数字，D3还支持格式化和解析日期，逗号分隔字串。

Numbers

d3.**format**(*specifier*)

返回给定的字符串（specifier）的格式化函数（等同于适用默认的美国英语语言环境的locale.numberFormat)。唯一的入参是数字，返回代表格式化数字的字符串。这个格式化规范模拟的是Python 3.1内置的格式化规范语言[[format specification mini-language|http://docs.python.org/release/3.1.3/library/string.html#formatspec]]。规范（specifier）的通常格式如下:

```
[[fill]align][sign][symbol][0][width][,][.precision][type]
```

*fill*可以是任意字符，除了 “{“ 和 “}”，fill 由紧跟它的*align*选项标识。

*align*有三种选项：

- ("<") 在可用的区域左对齐。
- (">") 在可用的区域右对齐（默认）。
- ("^") 在可用的区域居中。

*sign*可能是：

- plus ("+") - 可以用于正数或负数。
- minus ("-") - 仅仅用于负数（默认）。

- space (" ") - 前面的空格应该用在正数前面，而减号必须用在负数！

*symbol*可能是：

- currency ("$") - a currency symbol should be prefixed (or suffixed) per the locale.
- base ("#") - for binary, octal, or hexadecimal output, prefix by "0b", "0o", or "0x", respectively.

The "0" option enables zero-padding.

The *width* defines the minimum field width. If not specified, then the width will be determined by the content.

The *comma* (",") option enables the use of a comma for a thousands separator.

The *precision* indicates how many digits should be displayed after the decimal point for a value formatted with types "f" and "%", or before and after the decimal point for a value formatted with types "g", "r" and "p".

The available *type* values are:

- exponent ("e") - use [[Number.toExponential|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number/toExponential]].
- general ("g") - use [[Number.toPrecision|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number/toPrecision]].
- fixed ("f") - use [[Number.toFixed|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number/toFixed]].
- integer ("d") - use [[Number.toString|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number/toString]], but ignore any non-integer values.
- rounded ("r") - round to *precision* significant digits, padding with zeroes where necessary in similar fashion to fixed ("f"). If no *precision* is specified, falls back to general notation.
- percentage ("%") - like fixed, but multiply by 100 and suffix with "%".
- rounded percentage ("p") - like rounded, but multiply by 100 and suffix with "%".
- binary ("b") - outputs the number in base 2.
- octal ("o") - outputs the number in base 8.
- hexadecimal ("x") - outputs the number in base 16, using lower-case letters for the digits above 9.
- hexadecimal ("X") - outputs the number in base 16, using upper-case letters for the digits above 9.

- character ("c") - converts the integer to the corresponding unicode character before printing.
- SI-prefix ("s") - like rounded, but with a unit suffixed such as "9.5M" for mega, or "1.00µ" for micro.

The type "n" is also supported as shorthand for ",g".

d3.**formatPrefix**(*value*[, *precision*])

Returns the SI prefix for the specified *value*. If an optional *precision* is specified, the *value* is rounded accordingly using d3.round before computing the prefix. The returned prefix object has two properties:

- symbol - the prefix symbol, such as "M" for millions.
- scale - the scale function, for converting numbers to the appropriate prefixed scale.

For example:

```
var prefix = d3.formatPrefix(1.21e9);
console.log(prefix.symbol); // "G"
console.log(prefix.scale(1.21e9)); // 1.21
```

This method is used by d3.format for the `s` format.

d3.**round**(*x*[, *n*])

Returns the value *x* rounded to *n* digits after the decimal point. If *n* is omitted, it defaults to zero. The result is a number. Values are rounded to the closest multiple of 10 to the power minus *n*; if two multiples are equally close, the value is rounded up in accordance with the built-in
[[round|https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Math/round]]
function. For example:

```
d3.round(1.23); // 1
d3.round(1.23, 1); // 1.2
d3.round(1.25, 1); // 1.3
d3.round(12.5, 0); // 13
d3.round(12, -1); // 10
```

Note that the resulting number when converted to a string may be imprecise due to IEEE floating point precision; to format a number to a string with a fixed number of decimal points, use d3.format instead.

Strings

d3.**requote**(*string*)

Returns a quoted (escaped) version of the specified *string* such that the string may be embedded in a regular expression as a string literal.

```
d3.requote("[]"); // "\[\]"
```

Dates

See the [[d3.time|Time-Formatting]] module.

> Wiki ? [[API--中文手册]] ? [[核心函数]] ? *CSV格式化*

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

CSV 格式化 (d3.csv) ? d3.csv - 获取一个CSV (comma-separated values, 冒号分隔值)文件。 ? d3.csv.parse - 将CSV文件字符串转化成object的数组，object的key由第一行决定。如：[{"Year": "1997", "Length": "2.34"}, {"Year": "2000", "Length": "2.38"}] ? d3.csv.parseRows - 将CSV文件字符串转化成数组的数组。如： [["Year", "Length"],["1997", "2.34"],["2000", "2.38"]] ? d3.csv.format - 将object的数组转化成CSV文件字符串，是d3.csv.parse的逆操作。 ? d3.csv.formatRows - 将数组的数组转化成CSV文件字符串，是d3.csv.parseRows的逆操作。 ? d3.tsv - 获取一个TSV (tab-separated values, tab分隔值)文件。 ? d3.tsv.parse - 类似于d3.csv.parse。 ? d3.tsv.parseRows - 类似于d3.csv.parseRows。 ? d3.tsv.format - 类似于d3.csv.format。 ? d3.tsv.formatRows - 类似于d3.csv.formatRows。 ? d3.dsv - 创建一个类似于d3.csv的文件处理对象，可以自定义分隔符和mime type。如：vardsv = d3.dsv("|", "text/plain"); CSV D3解析comma-separated values，制表符分隔值和任意分割值提供内置支持。这些表格格式深受诸如Microsoft Excel电子表格程序。表格格式通常比JSON更省空间,这可以提高大型数据集加载时间。

d3.csv(url[, accessor][, callback])

在指定的url为逗号分隔值(CSV)文件发出一个HTTP GET请求。一般认为文件内容是RFC4180-compliant（应该是csv文件介绍）。请求的mime（多用途互联网邮件扩展类型）类型一般为“text/ csv”。此请求会当做异步处理,在打开请求后此方法会立即返回。CSV数据可用时,将以parsed rows作为参数调用指定的回调。如果出现错误,此回调函数将返回null。一个可选的访问器方法可能被指定,然后传递给d3.csv.parse;也可以通过使用返回请求对象的row函数来指定。例如: d3.csv("path/to/file.csv") .row(function(d){return{key:d.key,value:+d.value};}) .get(function(error,rows){console.log(rows);}); 查看样例：unemployment choropleth。

d3.csv.parse(string[, accessor])

通过一个CSV文件的内容解析指定的字符串，返回一个代表解析行的对象数组。一般认为文件内容是RFC4180-compliant。与parseRows方法不同的是,这种方法要求CSV文件的第一行包含一个以逗号分隔的列名;这些列名成为返回的对象的属性。例如,参考以下CSV文件: Year,Make,Model,Length 1997,Ford,E350,2.34 2000,Mercury,Cougar,2.38 生成的JavaScript数组: [{"Year":"1997","Make":"Ford","Model":"E350","Length":"2.34"}, {"Year":"2000","Make":"Mercury","Model":"Cougar","Length":"2.38"}] 值得注意的是这些值都是字符串；它们不会自动转为数字类型值。JavaScript会强制字符串自动转换成数字类型值(例如,使用+运算符)。通过指定一个访问器函数,您可以将字符串转换为数字或其他特定的类

型,如日期: d3.csv("example.csv",function(d){ return{ year:newDate(+d.Year,0,1),// convert "Year" column to Date make:d.Make, model:d.Model, length:+d.Length// convert "Length" column to number }; },function(error,rows){ console.log(rows); }); 尽管由很多的限制，但使用连接符”+”比parseInt或parseFloat 通常更快。例如,“30 px”当强制使用“+“返回NaN,而parseInt和parseFloat返回30。

d3.csv.parseRows(string[, accessor])

通过一个CSV文件的内容解析指定的字符串，返回一个代表解析行的对象数组。一般认为文件内容是RFC4180-compliant。与parse 方法,不论CSV文件是否不包含一个头，该方法将标题行作为标准并且使用。每一行都被表示为一个数组而不是一个对象。行可能会变长。例如,考虑以下CSV文件: 1997,Ford,E350,2.34 2000,Mercury,Cougar,2.38 生成的JavaScript数组: [["1997","Ford","E350","2.34"], ["2000","Mercury","Cougar","2.38"]] 值得注意的是这些值都是字符串；它们不会自动转为数字类型值。有关详细请参阅parse。 第二个参数（[, accessor]）可以指定一个访问器函数。这个函数调用CSV文件中的每一行数据，通过当前行数据对象和当前行索引作为两个参数。函数的返回值将取代所在返回数组里的元素数据；如果函数返回null，此行便从返回的数组里剔除。实际上，这个访问器类似于map 和filter操作符去返回数据行。访问器函数通过parse 将每一行转换为一个带有一些已命名属性的对象。

d3.csv.format(rows)

将指定数组里的行内容转换为逗号分隔值格式的字符串并返回。这个操作是parse方法的逆转。每一行将会由一个换行符(\ n)隔开,并在每一行的每一列将以逗号(,)隔开。数据值中包含的逗号，双引号(“)或换行符会使用双引号将其取代（最后这句实践后不理解，理解不透）。每一行视为一个对象,并且所有的对象属性将被转换成字段。为了更好的控制那些被转换的属性，将行内容转换为只包含这些应该被转换的属性的数组并且使用formatRows方法。

d3.csv.formatRows(rows)

将指定数组里的行内容转换为逗号分隔值格式的字符串并返回。这个操作是parseRows方法的逆转。每一行将会由一个换行符(\ n)隔开,并在每一行的每一列将以逗号(,)隔开。值所包含的逗号,双引号(“)或换行符会使用双引号将其脱逃。 TSV 除了分隔符由制表符代替了逗号（制表符分隔值相当于逗号分隔值）,其它没有太大区别。

d3.tsv(url[, accessor][, callback])

相当于d3.csv，只是分隔符为制表符而已。

d3.tsv.parse(string[, accessor])

相当于csv.parse，只是分隔符为制表符而已。

d3.tsv.parseRows(string[, accessor])

相当于csv.parseRows，只是分隔符为制表符而已。

d3.tsv.format(rows)

相当于csv.format，只是分隔符为制表符而已。

d3.tsv.formatRows(rows)

相当于csv.formatRows，只是分隔符为制表符而已。 Arbitrary Delimiters

d3.dsv(delimiter, mimeType)

对于给定分隔符和mime类型构造一个新的解析器。例如,解析值由“|”分隔,竖线字符,使用如下: vardsv=d3.dsv("|","text/plain");

dsv(url[, accessor][, callback])

相当于d3.csv，只是分隔符为具体值而已。

dsv.parse(string[, accessor])

相当于csv.parse，只是分隔符为具体值而已。

dsv.parseRows(string[, accessor])

相当于csv.parseRows，只是分隔符为具体值而已。

dsv.format(rows)

相当于csv.format，只是分隔符为具体值而已。

dsv.formatRows(rows)

相当于csv.formatRows，只是分隔符为具体值而已。

HarryT20140329

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 本地化

按语言和地区格式化数字、日期和不同的货币。D3默认的支持英语，你可以按照需要加载新的本地化来改变D3的格式化行为。

d3.**locale**(*definition*)

返回指定参数*definition*的新的本地化，数字格式的本地化必须包含下列属性：

- decimal - 数字区域字符串||（例如 `"."`）。
- thousands - 组分隔字符串（例如 `","`）。
- grouping - 分组大小数组（例如[3]），根据需要循环。
- currency - 货币前后缀字符串（例如 ["$", ""]）。

(注意：thousands 属性稍有命名不当，当组定义允许分组而不是几千。)

本地化定义必须包含以下时间属性：

- dateTime – 日期和时间(%c) 格式化字符串（例如："%a %b %e %X %Y"）。
- date - 日期 (%x) 格式化字符串（例如："%m/%d/%Y"）。
- time - 时间(%X) 格式化字符串（例如："%H:%M:%S"）。
- periods –本地的上午和下午，同样（例如：["AM", "PM"]）。
- days – 星期的全称，以Sunday开始。
- shortDays -星期的简称，以Sunday开始。
- months –月份的全称，以January开始。
- shortMonths -月份的简称，以January开始。

例如默认的美式英语 (en_US) 本地化定义为：

```
{
  "decimal": ".",
  "thousands": ",",
  "grouping": [3],
  "currency": ["$", ""],
  "dateTime": "%a %b %e %X %Y",
  "date": "%m/%d/%Y",
  "time": "%H:%M:%S",
  "periods": ["AM", "PM"],
  "days": ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"],
  "shortDays": ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"],
  "months": ["January", "February", "March", "April", "May", "June", "July", "August", "S
  "shortMonths": ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "
}
```

默认的俄语 (ru_RU) 本地化定义为：

```
{
  "decimal": ",",
  "thousands": "\xa0",
  "grouping": [3],
  "currency": ["", " руб."],
  "dateTime": "%A, %e %B %Y г. %X",
  "date": "%d.%m.%Y",
  "time": "%H:%M:%S",
  "periods": ["AM", "PM"],
  "days": ["воскресенье", "понедельник", "вторник", "среда", "четверг", "пятница", "суббо
  "shortDays": ["вс", "пн", "вт", "ср", "чт", "пт", "сб"],
  "months": ["января", "февраля", "марта", "апреля", "мая", "июня", "июля", "августа", "с
  "shortMonths": ["янв", "фев", "мар", "апр", "май", "июн", "июл", "авг", "сен", "окт", "
}
```

locale.**numberFormat**(*specifier*)

d3.format本地化。

locale.**timeFormat**(*specifier*)

d3.time.format本地化。

locale.timeFormat.**utc**(*specifier*)

d3.time.format.utc本地化。

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 颜色

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

实现可视化要经常和颜色打交道。虽然你的电脑显示屏懂得很多的颜色，但这对通过js来配置颜色帮助不大。所以d3提供多种颜色空间的表示，包括 RGB， HSL，LAB 和 HCL，可实现规范、插值、转换和操作（例如颜色的明暗）。 注意：虽然你可以直接操作颜色，当时也需要参考一下D3对于interpolateRgb, interpolateHsl 和 scales等内置颜色插值的实现。 如果你想查阅调色盘，请查阅ordinal scales 。

RGB

d3.rgb(r, g, b)

通过输入的参数r, g 和 b，创建一个RGB类型颜色对象。参数必须是在0-255之间的整数。你可以通过访问返回的颜色对象的r, g 和 b属性值来获取参数r, g, b的值。

d3.rgb(color)

• rgb decimal - "rgb(255,255,255)" • hsl decimal - "hsl(120,50%,20%)" • rgb hexadecimal - "#ffeeaa" • rgb shorthand hexadecimal - "#fea" • named - "red", "white", "blue"

通过解析输入的字符串参数，创建一个RGB类型颜色对象。如果参数 color 不是一个字符串，参数就会被强制类型转换为字符串类型。因此，该构造函数可以用来创建一个已经存在的颜色对象的副本，或者是将d3.hsl强制类型转换为RGB。字符串参数可以有多种形式： • rgb十进制- "rgb(255,255,255)" • hsl十进制- "hsl(120,50%,20%)" • rgb十六进制- "#ffeeaa" • rgb简写十六进制- "#fea" • 名称 - "red", "white", "blue" 输出的颜色将以红、绿和蓝的整数通道形式存储（整数范围：[0,255]）。颜色通道可以通过颜色对象的属性 r, g 和 b 访问到。可支持的列表named colors可以通过CSS指定。如果是HSL空间的颜色，可以转换成在RGB空间相同类型的值，和 hsl.rgb类似。

rgb.brighter([k])

返回颜色的一个高亮副本。每个颜色通道值将乘以0.7 ^ -k. 如果参数 k 被忽略，将使用默认值1。通道值上限值255，下限值30.

rgb.darker([k])

返回低颜色的一个亮度副本。每个颜色通道值将乘以0.7 ^ k. 如果参数 k 被忽略，将使用默认值1。

rgb.hsl()

返回一个HSL空间的等值颜色对象。请查阅 d3.hsl 了解更多关于返回颜色对象的信息. 文档CSS3 Color Module Level 3中有关于RGB到HSL转换的信息。该函数是上述转化的逆操作。

rgb.toString()

将RGB颜色转换成一个十六进制数的字符串，如 such as "#f7eaba"。

HSL

d3.hsl(h, s, l)

通过输入的参数h, s和l，创建一个HSL颜色对象。创建新的HSL颜色，通过指定的色度h，饱和度s和亮度l。其中色度h取值范围[0,360]。饱和度和亮度取值范围 0,1。你可以通过访问返回的颜色对象的h, s 和l 属性值来获取颜色的通道属性.

d3.hsl(color)

• rgb decimal - "rgb(255,255,255)" • hsl decimal - "hsl(120,50%,20%)" • rgb hexadecimal - "#ffeeaa" • rgb shorthand hexadecimal - "#fea" • named - "red", "white", "blue"

通过解析输入的字符串参数，创建一个HSL类型颜色对象。如果参数 color 不是一个字符串，参数就会被强制类型转换为字符串类型。因此，该构造函数可以用来创建一个已经存在的颜色对象的副本，或者是将d3.rgb强制类型转换为HSL。字符串参数可以有多种形式：• rgb十进制- "rgb(255,255,255)" • hsl十进制- "hsl(120,50%,20%)" • rgb十六进制- "#ffeeaa" • rgb简写十六进制- "#fea" • 名称 - "red", "white", "blue" 输出的颜色将以取值范围为[0,360]的色度和取值范围为[0,1]的亮度、饱和度作为属性值存储。红、绿和蓝的整数通道形式存储（整数范围：[0,255]）。颜色通道可以通过颜色对象的属性h, s和l访问到。可支持的列表named colors可以通过CSS指定。如果是RGB空间的颜色，可以转换成在HSL空间相同类型的值，和rgb.hsl类似。

hsl.brighter([k])

返回颜色的一个高亮副本。每个颜色通道值将乘以0.7 ^ -k. 如果参数 k 被忽略，将使用默认值1。通道值上限值255，下限值30.

hsl.darker([k])

返回低颜色的一个亮度副本。每个颜色通道值将乘以0.7 ^ k. 如果参数 k 被忽略，将使用默认值1。

hsl.rgb()

返回一个RGB空间的等值颜色对象。请查阅d3.rgb了解更多关于返回颜色对象的信息. 文档CSS3 Color Module Level 3中有关于HSL到RGB 转换的信息。该函数是上述转化的逆操作。

hsl.toString()

将颜色转换成一个十六进制数的RGB颜色字符串，如 such as "#f7eaba"。

HCL

d3.hcl(h, c, l)

…

d3.hcl(color)

…

hcl.brighter([k])

…

hcl.darker([k])

…

hcl.rgb()

…

hcl.toString()

将颜色转换成一个十六进制数的RGB颜色字符串，如 such as "#f7eaba"。 Lab*

d3.lab(l, a, b)

…

d3.lab(color)

…

lab.brighter([k])

…

lab.darker([k])

…

lab.rgb()

…

lab.toString()

将这个Lab*颜色转换成一个十六进制数的RGB颜色字符串，如 such as "#f7eaba"。

Color

提供d3.color的基础类型，支持你扩展D3增加行的颜色空间。这个类型能够自动地使用d3.interpolate（通过instanceof d3.color识别）插入RGB。

d3.color()

颜色类型的基础构造函数。

color.rgb()

返回这个颜色的RGB值，必须被所有的颜色空间实现。

color.toString()

转换为代表这个颜色的RGB十六进制字符串，例如"#f7eaba"。 边城译2014-4-6 咕噜校20141122

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 命名空间

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

Namespaces Wiki ▸ API Reference ▸ Core ▸ Namespaces SVG具有各种不同来自HTML的命名空间，所以D3提供了一些工具来简化处理命名空间。

d3.ns.prefix

注册的命名空间前缀map如下图。默认值是：

{ svg:"http://www.w3.org/2000/svg", xhtml:"http://www.w3.org/1999/xhtml", xlink:"http://www.w3.org/1999/xlink", xml:"http://www.w3.org/XML/1998/namespace", xmlns:"http://www.w3.org/2000/xmlns/" } 当需要创建其他命名空间中的元素或属性时可以指定附加前缀。

d3.ns.qualify(name)

符合规定的名称，可能有一个命名空间前缀。如果名字包含冒号(":")，在冒号前的子字符串是可以解释为命名空间前缀，它必须注册在d3.ns.prefix中；返回值是一个对象，带有的space和local 属性，含有完整命名空间URL和本地名称。例如，("svg:text")合格的结果是：

{space:"http://www.w3.org/2000/svg",local:"text"} 如果名称不包含冒号，这个函数只返回输入名称。此功能是在内部使用，以决定是否使用名称空间的方法（如createElementNS）或一个非命名空间的对等物。

马语者T20140405_guluP20140405

> Wiki ▸ [[API--中文手册]] ▸ [[核心函数]] ▸ 内部

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

实施可重用组件的各种工具。 函数

d3.functor(value)

如果参数value 是个函数，返回这个函数。否则，返回一个能够输出这个参数的函数变量。该方法用来将常量参数升级转换成函数，以备需要指定属性为常量或者函数的时候，直接实现。比如：许多D3 layouts需要指定属性成这种格式，当我们自动转换值到函数的时候，这样可以简化实现。

d3.rebind(target, source, names…)

将方法从指定的参数source拷贝到target。当调用target，将相当于调用函数source。注意，传递到target的参数，将传递到source中。target使用source作为this的上下文。如果source返回了source对象，那么相应的target将会返回target对象。否则，target返回source返回的值。方法rebind 允许继承的方法绑定到一个不同对象的子类。

Events 事件

D3中的行为和高级组件，如 brush，使用d3.dispatch 来传递事件消息。

对于多关联视图的可视化，d3.dispatch提供一个方便的轻量级的机制来处理相关联的组件。将代码和d3.dispatch结合起来，可将涉及的多个事件分离，更好的维护自己的代码。

d3.dispatch(types…)

为指定的types创建一个dispatcher对象。每个字符串参数表示一个事件相应，比如："zoom" 和 "change"。返回的对象是一个关联的数组。每个type和一个dispatch 相关联。如果你想为start和end创建一个event dispatcher， 可以这样： vardispatch=d3.dispatch("start","end"); 然后，你可以访问dispatchers的属性来获取不同的事件相应属性： dispatch.start and dispatch.end. 如，你可以添加一个事件的监听： dispatch.on("start",listener); 然后传递事件到所有注册的监听器上： dispatch.start(); 关于如何实现将参数传递到监听器，详见：dispatch。

dispatch.on(type[, listener])

为指定的type添加或删除一个事件监听。其中 type 是一个事件名，如"start" 或 "end"。函数调用将参数和上下文传递给监听，并触发监听。详见 dispatch. 如果出在事件监听注册了某个type，已经存在的监听将被删除，然后注册新的监听。为了注册多个事件监听到同一个type，可以为这个typy提供命名空间，如： "click.foo" 和 "click.bar"。 如果参数中没有监听，则默认为给指定的type设置当前的监听。

dispatch.type(arguments…)

type 方法 (如上文中的 dispatch.start) 通知并将参数传递给注册的监听。上下文 this 作为注册监听的上下文。例如：通过foo 和bar参数值触发所有的监听，比如dispatch.call(foo, bar)。因此，你可以传递任何参数到指定的监听器上。通常，我们通过创建一个对象来表示一个事件相应，或者是传递当前的datum (d) 和 index (i)。也可以使用 call 或者 apply来设置监听器的"this" 上下文。 举例说明：如果想要为"custom" 事件添加一个"click" 事件，并预置上下文this和参数： selection.on("click",function(d,i){ dispatch.custom.apply(this,arguments); });

边城T20140403_guluP20141122

> Wiki ▸ [[API--中文手册]] ▸ 比例尺

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

See one of:

- [[数值比例尺]] - 定义域是连续的，例如数字。
- [[序数比例尺]] - 定义域是离散的，例如名称或类别。
- [[时间比例尺]] - 定义域是时间。

> Wiki ▸ [[API--中文手册]] ▸ [[比例尺]] ▸ 数值比例尺

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

Linear Scales 线性比例尺是最常见的比例尺，为连续地把输入域映射到连续的输出范围提供了良好的缺省选择。该映射是线性的，输出范围值y可以表示为输入域值x的线性函数为：y = mx + b。输入域通常是要可视化的数据维度，如学生在样本群的身高（米为单位）。输出范围通常是所需输出的可视化维度，如直方图的中条的高度（以像素为单位）。

d3.scale.linear()

用默认域[0,1]构造一个新的比例尺，默认的范围为[0,1]。因此，默认比例尺相当于数字恒等函数；例如linear(0.5)返回0.5。

linear(x)

在输入域中的输入x，返回输出范围对应的值。 注意：某些插值器会重用返回值。例如，如果域值是任意的对象，然后d3.interpolateObject自动执行，比例尺重用返回的对象。通常情况下，一个比例尺的返回值被立即用于设置属性或样式，你不必担心这一点；但是，如果你需要存储比例尺的返回值，使用字符串强制转换或酌情创建一个副本。

linear.invert(y)

返回值的输入域X在输出范围y中的相应值。这代表了逆映射的范围域。在输出范围的有效值y，线性（ linear.invert （ Y） ） 等于y ;类似地，在输入域中的有效数值x， linear.invert （线性（X））等于x时。等价地，你可以通过建立一个新的比例尺，交换定义域和值域进行反转操作。翻转操作在交互中特别有用的，例如，以确定在对应于该像素位置的鼠标下的输入域的值。 注：如果输出范围是数字只能用倒置操作来支持！ D3允许的输出范围是任何类型的;引擎盖下， d3.interpolate或您选择的自定义插值是用来归一化参数t映射到输出范围内的值。因此，输出范围可以是颜色，字符串，或者甚至任意对象。由于没有设施，以“ uninterpolate ”来标注任意类型，倒置操作目前仅在数值范围的支持。

linear.domain([numbers])

如果数字被指定，设置比例尺的输入域到数字的指定数组。数组必须包含两个或两个以上的数字。如果给定的数组中的元素不是数字，他们将被强制转换为数字;这种强迫发生同样，当规模被调用。因此，线性标尺可用于类型，如日期的对象可以被转换成数字编码;但是，它通常是用d3.time.scale的日期更方便。 （您可以使用的valueOf实现自己的转换数量的对象。）如果未指定数字，则返回比例尺的输入域。 虽然线性标度通常只有两个数值在其域中，可以用polylinear比例尺指定两个以上的值。在这种情况下，必须有值的输出范围内的当量数。一个polylinear刻度表示多个分段线性尺度上划分一个连续的定义域和值域。这是用于定义发散的定量尺度是特别有用的。例如，把白色和红色作为负值，白色和绿色作为正值：

所得到的值的颜色（ - 0.5）是RGB（255，128，128）和色彩（0.5）的值是RGB（128，192，128）。在内部，polylinear比例尺通过二分查找对应于给定域值输出插值。通过重复在这两个领域和范围值，你也可以强制输入域的块映射到一个固定的输出范围。

linear.range([values])

如果值被指定，设置刻度的输出范围值的指定数组。数组必须包含两个或多个值，以匹配输入域的基数，否则第二个的长被截断以匹配另一个。给定的数组中的元素不必是数字;所支持的底层的内插器的任何值都可以。然而，数值范围是必需的倒置运算符。如果未指定值，则返回当前比例次的输出范围。

linear.rangeRound(values)

用指定数组设置比例尺的范围值，而比例尺的插补器也设置为d3.interpolateRound。这是用于当输出由刻度值应该是准确的整数，例如，以避免抗混叠伪像一个方便例程。另外，也可以在比例被施加之后使用round操作。

linear.interpolate([factory])

如果指定了工厂，设置使用指定的工厂规模的输出插值。内插器工厂默认为d3.interpolate，并用于在[0,1]范围内的归一化域参数t在输出范围对应的值。内插器工厂将用于从输出范围构造内插器为每一对相邻的值。如果未指定工厂，返回规模的插补工厂。

linear.clamp([boolean])

如果布尔变量被指定，用于启用或禁用相应地夹集。默认情况下，夹集被禁用，例如，如果输入域之外的值传递给规模，规模可能外面的输出范围，通过线性外推法返回一个值。例如，使用缺省域并[0,1]范围内，与2的输入值将返回2的输出值，如果夹集被启用，归一化的

域参数t被钳位到范围[0,1]这样规模的返回值总是规模的输出范围之内。如果未指定布尔值，则返回的规模当前夹值输出范围之内。

linear.nice([count])

它开始和结束圆标记值的扩展域。这种方法通常会修改规模域，并且只能在边界延伸到最近的整数值。根据下列公式计算圆值的精度依赖于域dx的程度：EXP（圆形（日志（DX））-1）。如果该域是从数据计算并可以是不规则，则Nicing是有用的。例如，对于[0.20147987687960267，0.996679553296417]一个领域，漂亮的论域为[0.2，1]。如果域有两个以上的值，nicing域只影响第一个和最后一个值。 可选的计数参数允许更大的控制权用于扩展边界的步长，保证返回域可完全覆盖域。

linear.ticks([count])

从比例尺上的输入域的代表值返回约数。如果未指定count，则默认为10。返回的刻度值是均匀间隔的，具有人类可读的值（如10的幂的倍数），并且保证是输入域的范围之内。刻度尺通常用于显示参考线或刻度线，在与可视化数据一起使用。指定的计数是只是一个提示;规模可能取决于输入域返回更多或更少的值。

linear.tickFormat(count, [format])

返回格式化的数字用于显示刻度值的值。指定的计数应具有相同的值被用于产生刻度的计数。您不必使用刻度尺的内置刻度格式，但它会自动计算的基础上刻度值之间的固定间隔适当的精度。 可选的格式参数允许指定的格式说明符，其中格式的精度自动被取代的规模是适当的刻度间隔。例如，要格式化百分比变化，你可能会说：

如果格式已经指定了精度，这种方法相当于d3.format。

linear.copy()

返回该线性标尺的精确副本。改变这一比例尺不会影响返回的比例尺，反之亦然。 Identity Scales 恒等尺是线性尺的一个特例其定义域和值域相同；尺以及它的反转方法都是恒等的。与像素坐标工作时，这些尺度偶尔有用，例如在与轴axis和刷brush组件一起使用。

d3.scale.identity()

构造一个新恒等尺，默认定义域[0，1]，默认值域是[0，1]。恒等尺总是等同于恒等函数。

identity(x)

identity.invert(x)

返回给定值x。

identity.domain([numbers])

identity.range([numbers])

如果指定了数字，设置尺度的输入域和输出范围为指定的数字数组。这个数组必须包含两个或两个以上的数字。如果给定的数组中的元素不是数字，他们将被强制转换为数字; 这种强制转换在尺scale被调用时同样发生。 如果未指定数字，则返回尺的当前输入定义域（或等价地，输出范围）。

identity.ticks([count])

大致返回刻度的输入域（或等价地，输出范围）count 代表性的值。如果未指定count，则默认为10。返回的刻度值是均匀间隔的，具有人类可读的值（如10的幂的倍数），并且保证是输入域的范围之内。刻度线通常用于显示在与可视化数据一起使用的参考线或刻度线。指定的计数是只是一个提示;刻度取决于输入域可能返回更多或更少的值。

identity.tickFormat(count, [format])

返回适合用于显示刻度值的数字格式number format函数。指定的计数count 应和用于产生刻度值的计数的值相等。您不必使用尺度的内置刻度格式，但它会基于刻度值之间的固定间隔自动计算适当的精度。 可选参数format 允许指定格式说明符。如果格式说明符没有一个定义的精度，精度将由尺自动设置，并返回相应的格式。这提供了方便，可用声明的方式指定一个格式其精度将由尺度自动设置。

identity.copy()

返回此尺度的精确副本。改变这一尺度不会影响返回的尺度，反之亦然。

Power Scales 除了有一个在输出范围值被计算之前应用于输入值域的指数变换之外，乘方比例尺类似于线性比例尺。映射到输出范围值y可以被表示为一个输入值域x的函数：y = mx ^ k + b,其中k是指数的值。乘方比例尺也支持负值，在这种情况下，输入值权力天平也支持负值,在这种情况下,输入值以指数-1进行乘方计算,由此产生的输出值也将以-1来乘方计算。

d3.scale.sqrt()

构造一个新的乘方比例尺，输入值域默认为[0,1],输出范围默认为[0,1],指数默认为0.5。这种方法可以简记为: d3.scale.pow().exponent(.5) 返回的比例尺是一个函数,它接受一个代表输入值域中某一个值的参数x,返回值是对应于输出范围的一个值。因此,这个比例尺对于数字来说，相当于函数sqrt的功能；例如:sqrt(0.25)的结果将返回0.5。

d3.scale.pow()

构造一个新的乘方比例尺，默认输入值域为[0,1],默认输出范围[0,1],默认指数为1。因此,对于数字来说，默认的乘方比例尺相当于数字正比例函数（the identity function）;例如pow(0.5)的结果将返回0.5。

pow(x)

给定一个输入域值x,返回输出范围中相应的值。 注意:一些interpolators的重用将返回值。例如,如果值域是任意对象,然后d3.interpolateObject自动被应用并且比例尺重用这个返回的对象。通常,一个比例尺的返回值会立刻被应用于设置一个属性(attribute)或样式(style),你不必担心；不过,如果你需要存储比例尺的返回值,可以适当地使用字符串强转换或复制值。

pow.invert(y)

返回对应于输出域值y值的输入值域值x。这意味着从输出范围到输入值域映射关系的逆向操作。对于一个有效的输出范围值y，pow(pow.invert(y))的结果等于y， 对于一个有效的输入域值x，pow.invert(pow(x))的结果等于x。同样，你可以通过交换输出与输入值域的新比例尺的建立去创建其逆操作。对于确定输入值域的值（此值对应于鼠标下的像素位置）而言，这个逆操作对于交互尤为重要。

注意:逆操作仅仅支持输出范围是数字的情况！D3是允许输出范围属于任意类型；在其内部结构中，d3.interpolate或者一个自定义的插值器是用来映射这个标准化参数t到一个输出范围的值。因此，输出范围可能是颜色值、字符串、甚至任意的对象。 转化运算符仅支持如果输出

范围是数字!D3允许任何类型的输出范围;在引擎盖下,d3.interpolate或您选择的自定义插入器是用来映射值的归一化参数t的输出范围。因此,输出范围可能是颜色,字符串,甚至任意对象。所以对于“非差值”的任意类型是没有任何措施的，逆操作目前仅仅支持数字范围。

pow.domain([numbers])

如果参数numbers被指定，便将指定包含numbers的数组设置为比例尺的输入值域。这个数组必须包含2个或更多的数据值。如果给定的数组中的元素不是数字，它们便被强转为数字；当比例尺被调用时强转同样会发生。因此，一个乘方比例尺被用来编码可以被转换为数字的任意类型。如果numbers没有被指定，将返回比例尺当前的输入值域。 与线性比例尺(详见linear.domain)一样,对于输入和输出值域乘方比例尺也可以接受超过两个以上的值,因而产生结果的polypower比例尺。# pow.range([values]) 如果values被指定,便将指定包含values的数组设置为比例尺的输出范围。这个数组必须包含两个或两个以上的值,来对应匹配输入域的基数,否则多于两个以上的将被截断从而去对应其它值。给定数组中的元素不要求必须为数字;支持基本插值器的任何值都可以使用。然而,数字范围需要转化操作符去转化。如果values没有被指定,将返回当前的输出范围。

pow.rangeRound(values)

将指定的values组成的数组设置为输出范围，同时设置比例尺的插值器到d3.interpolateRound。当比例尺输出的值需要为整数时，这便是一个非常方便使用套路，以此避免不确定性的小数产生。也可以在比例尺应用后通过手动将输出值处理为整数。

pow.exponent([k])

如果k值被指定值,当前指数将被设置为k值。如果k值没有被指定,则返回当前指数。默认值是1。

pow.interpolate([factory])

如果工厂（interpolator）被指定,便将此指定的工厂设置为比例尺的输出插值器。插值器工厂默认用d3.interpolate,并且它被用来将标准的[0,1]的值域参数t映射到相关的输出值域中对应的值。这个插值器工厂将针对输出值域中每两个相邻的值来用于构建插值器。如果没有指定工厂,将返回比例尺的插入器工厂。

pow.clamp([boolean])

如果布尔值（boolean）被指定,启用或禁用相应的闭合。默认情况下,是启用闭合,这样的情况下,如果一个超出输入值域的值在传递给比例尺时，比例尺将通过线性外推法返回一个输出范围外的一个值。例如,使用输入和输出值域采用默认值域[0,1],那么对于输入值为2的就会返回一个输出值2。如果启用了闭合,标准的值域参数t将被闭合在值域区间[0,1]中，这样的情况下，比例尺的返回值总是在比例尺的输出范围内。如果布尔值不被指定,返回值都将在输出范围内（returns whether or not the scale currently clamps values to within the output range）。

pow.nice([m])

扩展值域,即它的开始和结束值都处理为容易识别的整数值（Extends the domain so that it starts and ends on nice round values）。这种方法通常来修改比例尺的值域,可能只是将临界的值处理为最近的整数值（This method typically modifies the scale's domain, and may only extend the bounds to the nearest round value）。这个整数值的精度依赖于值域dx的范围，根据下列的公式：exp(round(log(dx)) - 1)。如果值域是通过不规则的值计算而来的，那么这样处理（使用本函数）是很有用的。例如,对于一个值域为[0.20147987687960267,0.9966795532964171],那么好的值域（可以通过本函数处理）便是(0.2,1)。如果值域拥有超过两个的值,那么这样处理后只能影响到第一个和最后一个值。 可选参数m允许指定一个滴答计数用来被指定用来在扩展边界值之前去控制步长使用。

pow.ticks([count])

近似地返回几个（count：个数）来自于比例尺输入值域的代表数。如果count没有被指定,默认为10。返回的值都有均匀一致的间隔,拥有人类可读的值(如乘方指数为10的倍数(such as multiples of powers of 10)),并保证是在输入值域的范围内。刻度值（Ticks）通常用于结合可视化数据去显示参考线,或刻度线。指定的数量只是一个暗示；比例尺可能会返回值或多或少，这取决于输入值域。

pow.tickFormat([count, [format]])

返回一个数字格式化（number format）函数，适用于显示一个点值。被指定的count数量值与被用来生刻度值的数量相同。你不必使用比例尺的内置的刻度值格式化,但它能够基于固定时间间隔值去自动计算出适当的精度。 可选参数format允许为被指定的格式说明符（format specifier）。如果格式说明符没有明确定义的精度,那么精度将通过比例尺被自动设置，返回适当的格式。这提供了一个便捷的、声明的方式去指定一个格式，此格式的精度将通过比例尺被自动设定。

pow.copy()

返回一个比例尺的精确复制体。这个比例尺的变化不会影响到返回的复制体比例尺,反之亦然。

Log Scales 对数比例尺类似于线性比例尺，除了有一个对数变换被应用于在输出范围值计算的输入值域的值。映射到输出范围的y值可以表示为一个输入值域x的函数： y = m log(x) + b。 log(0)是负无穷大，一个对数比例尺必须有一个专门的正域或负域；这个范围不能包含或过零。一个正域对数比例尺有一个定义明确的正值的行为状态，一个负域对数比例尺有一个定义明确的负值的行为状态（输入值乘以-1，所得到的输出值也乘以-1）。如果你传递一个负值给对数比例尺是正域，反之比例尺的行为是未定义的。

d3.scale.log()

构建一个默认域[1,10]新的对数比例尺，默认范围[0,1]，底数是10。

log(x)

给一个在输入值域中的x值，返回一个输出范围的相应值。 注：一些内插重用的返回值。例如，如果域的值是任意对象，那么d3.interpolateobject是自动应用和按比例返回的对象。通常情况下，一个比例尺的返回值被立即用于设置属性或样式，你不必担心这一点;但是，如果你需要存储比列尺的返回值，使用字符串控制或创建一个合适的副本。

log.invert(y)

返回值的输入值域X在输出范围y中的相应值。这代表了逆映射的范围域。在输出范围有效的y值，log(log.invert(y))等于y;同样，在输入值域中的有效值x，log.invert(log(x)) 等于x。相同地，你可以通过建立一个新的比例尺，对变换输入值域和输出范围进行反转操作。反转操作是交互特别有用的，例如，以确定在对应于该像素位置的鼠标下的输入值域的值。 注：反转操作仅支持输出范围是数字！ D3允许的输出范围是任何类型;在后台，d3.interpolate或您选择的自定义插值是用来归一化参数t映射到输出范围内的值。因此，输出范围可以是颜色，字符串，或者甚至任意对象。由于没有便利以“篡改”任意类型，反转操作目前仅在数值范围的支持。

log.domain([numbers])

如果指定的数字，设置比例尺的输入值域给数字的指定数组。数组必须包含两个或两个以上的数字。如果给定数组中的元素不是数字，他们将被强制转换为数字;这种强制同样发生在当比例尺被调用。因此，一个对数比例尺可用于能被转换成数字的任何类型。如果数字未指定，则返回比例的当前输入值域。 如线性比例尺（见linear.domain），对数比例尺也可以接受两个以上值的值域和范围，从而得到聚对数函数。

log.range([values])

如果值是指定的，要通过指定数组的值设置比例尺的输出范围。数组必须包含两个或多个值，以匹配输入值域的底数，否则两个中长的会被截断以匹配另一个。给定的数组中的元素不必是数字;所支持的底层的内插器的任何值都可以。然而，数值范围是必需的倒置运算符。如果未指定值，则返回比例尺的当前输出范围。

log.rangeRound(values)

设置比例尺的输出范围要通过指定数组的值，而比例尺的内插器也设置为d3.interpolateRound。这是用于输出的比例尺的值应该是准确的整数，以避免抗混叠伪像的一个方便例程。另外，当输出值的比例尺被应用之后也可以手动设置四舍五入求值。

log.base([base])

如果指定了底数，设置这个对数比例尺的底数。如果未指定底数，返回当前的底数，默认为10。

log.interpolate([factory])

如果工厂模式是指定的，设置比例尺的输出内插器使用指定的工厂模式，内插器的工厂模式默认为d3.interpolate，并用于在[0,1]的归一化域参数t映射到输出范围对应的值。内插器工厂模式将用于从输出范围创建内插器为每一对相邻的值，如果没有指定的工厂模式时，返回比例尺的内插器的工厂模式。

log.clamp([boolean])

如果布尔值是指定的，启用或禁用相应的锁定。默认情况下，锁定被禁用，例如，如果输入值域之外的值传递给比例尺，比例尺可能返回外面的输出范围，通过线性外推法返回一个值。例如，使用默认值域并在[0,1]范围内，一个2的输入值将返回2的输出值，如果锁定被启

用，归一化的域参数t被锁定到范围[0,1]，这样比例尺的返回值始终在比例尺的输出范围之内。如果未指定布尔值，则返回比例尺当前锁定值是否在输出范围内。

log.nice()

扩展值域以便它开始和结束美化四舍五入的值。这种方法通常会修改比例尺的值域，并可能只有边界延伸至最近的四舍五入的值。最近的整数值是基于比例尺的底数的整数幂，默认为10。美化是有用的，如果该值域是从数据计算的并可以是不规则的。例如，对于值域[0.20147987687960267，0.9966795532964171]，美化的值域为[0.1，1]。如果值域有两个以上的值，美化值域只影响第一个和最后一个值。

log.ticks()

从比例尺上的输入值域将返回代表值。返回的刻度值是均匀间隔在10的每个幂的范围内，并保证是输入值域的范围之内。刻度通常用于显示参考线或刻度线，与可视化数据一起使用。请注意，刻度的数量不能自定义（由于对数比例尺的性质）;但是，如果你想减少刻度的数量，你可以筛选返回的数组中的值。

log.tickFormat([count, [format]])

返回一个数字格式的函数适合用于显示一个刻度值。这个返回刻度格式是为实现d.toPrecision（1）。如果一个计数被指定，那么一些刻度标记可能无法显示; 如果没有足够的空间来容纳所有的刻度标记，这是非常有用的。但是，请注意刻度线仍然会显示出来（这样对数比例尺失真仍然可见）。当指定一个计数，你也可能会覆盖这个格式函数; 您也可以指定一个格式说明符的字符串，它会自动被包裹在d3.format。例如，要获得一个刻度格式，将显示20个刻度的货币： scale.tickFormat(20, "$,.2f"); 如果格式说明符没有一个定义的精度，精度将通过比例尺自动设置，返回适当的格式。这提供了一个便利，既指定其精度格式的声明方式将由比例尺自动设定。

log.copy()

返回此比例尺的精确副本。改变这一比例尺不会影响返回的比例尺，反之亦然。

Quantize Scales

d3.scale.quantize()

使用默认的定义域[0,1]，定义一个量化变换，默认的范围是[0,1]；因此，默认的量化变换等同于数值的四舍五入round 方法；例如：quantize(0.49)返回0，quantize(0.51)返回1，详见下例： var q = d3.scale.quantize().domain([0, 1]).range(['a', 'b', 'c']); //q(0.3) === 'a', q(0.4) === 'b', q(0.6) === 'b', q(0.7) ==='c'; //q.invertExtent('a') returns [0, 0.3333333333333333]

quantize(x)

指定一个值x作为输入域值，返回对应该域值的范围值； 理解：quantize(0.4)则会使用默认的值域0和1，即：该quantize(0.4)返回值为0，因为四舍五入0.4为0，以此类推；

quantize.invertExtent(y)

根据输出的范围值y返回对应于定义域中的取值范围，即为数组[x0, x1]，该功能即：反向找到值y对应的定义域；这是非常有用的，例如：为了确定对应于该像素的鼠标的可取输入域；

quantize.domain([numbers])

如果numbers指定，则设置变换的定义域/输入域为numbers，该numbers是两个元素的数组，即代表定义域的开始和结束；如果该数组的元素多余2个数值，则只会去第一个元素为开始，最后一个元素为结束；如果该数组中元素不是数值，则会强制转换为数值；强制转换会在调用该scale 转换时发生；因此，定量变换可被用于任何可强转成数值的编码/参数类型；如果numbers未指定，则返回当前的定义域/输入域；

quantize.range([values])

如果values 指定，则设置变换的值域/输出域为values，该values 可以是任何类型的数组元素，并且元素的数量不限；如果values 未指定，则返回当前的值域/输出域；

quantize.copy()

返回当前变换的一个副本，无论是改变原本或副本，都不会影响彼此；

Quantile Scales 分位数比例尺映射输入域到离散的的范围。即使输入域是连续的比例尺接受任意合理的输入值，输入域指定为一组离散的值。输出范围的值的数量（基数）决定输入域中将被计算的分位数数量。为了计算分位数，输入域是排好序的，并且被当成离散值的群体。输入域通常是你想要可视化的数据维度，例如股票市场每天的变化。输出范围通常是需要输出的可视化，例如一个发散的颜色尺。

d3.scale.quantile()

构造一个新的分位数比例尺，使用空的输入域和输出范围。分位数比例尺在输入域和输出范围没有同时指定时无效。

quantile(x)

给定输入域中的值x，返回输出范围中对应的值。

quantile.invertExtent(y)

为对应输出范围中的值y，返回输入域[x0, x1]的范围，代表从范围到域逆向映射。这个方法在交互时是很有用的。用来决定输入域中的值对应鼠标下的像素位置。

quantile.domain([numbers])

如果指定了参数numbers，设置分位数比例尺的输入域为指定的离散数值集。数组必须是非空的，并且必须包含至少一个数值；NaN，null和未定义值是被忽略的，不被视为样本群体的一部分。如果给定数组中的元素不是数字，将被强制转换为数字；当比例尺被调用的时候强制转换。这样，分位数比例尺可以用来强制转换所有可以转换为数字的类型。如果numbers参数没有指定，就返回比例尺当前的输入。

quantile.range([values])

如果指定了values参数，就设置输出范围中的离散值。数组必须是非空的，可能包含任意类型的值。Values数组中值得数量（基数或长度）决定了将被计算的分位数的数量。例如，为了计算分位数，values必须是一个像[0, 1, 2, 3] 的四元素的数组。如果没有指定values参数，返回当前的输出范围。

quantile.quantiles()

返回分位数阈值。如果输出范围包含n个离散值，返回的阈值数组就包含n-1个值。值小于阈值数组的第一个元素，quantiles()[0]视为在第一个分位；更大的值第二个阈值在第二个分位，等等。在内部，阈值数组用于d3.bisect 查找给定输入值对应的输出分位。

quantile.copy()

返回这个比例尺的一个精确的副本。比例尺的改变不影响返回的比例尺，反之亦然。

Threshold Scales 临界值比例尺很像定量 quantize比例尺，只是他允许你将输入域的任意的子集合映射到离散的输出值上。输入值是连续的，并且基于一组临界值被划到不同的区域。输入值域通常是你想要用来可视化的数据的尺度，例如是一群学生的身高。输出值域通常是所需的输出可视化数据的尺度，。例如是一组颜色（用字符串表示）。

d3.scale.threshold()

构造一组新的临界值比例尺，默认的输入范围是[.5]，默认的输出范围是[0,1]，因此，对数字来说，默认quantize比例尺将和round函数相同，例如，输入0.49将输出0，输入0.51将输出1. var t = d3.scale.threshold().domain([0, 1]).range(['a', 'b', 'c']); t(-1) === 'a'; t(0) === 'b'; t(0.5) === 'b'; t(1) === 'c'; t(1000) === 'c'; t.invertExtent('a'); //returns [undefined, 0] t.invertExtent('b'); //returns [0, 1] t.invertExtent('c'); //returns [1, undefined]

threshold(x)

x代表输入域值，函数将返回对应的输出范围的值。

threshold.invertExtent(y)

返回输出范围y中值对应的输入域值的范围，代表输出值到输入值的一个反向映射，这个方法对于交互非常有用，例如可以用来定位鼠标的像素点所在的区域。

threshold.domain([domain])

如果指定了domain参数，那么将这个数组形式的参数作为输入域，参数数组的值必须是升序排列的，否则比例尺将是不确定的，这个参数数组的值通常是数字，但是任何自然排序的值（例如字符串）都可以，因此一个临界值比例尺适用于任何可以排序的类型。如果输出范围是N+1那么输入值范围一定是N，如果小于N的元素在输入值里，那么这个额外的元素将被忽略，如果大于N的元素在输入值里，那么对于一些输入值比例尺将返回未定义，如果输入域未定义，将返回这个比例尺当前的输入值

threshold.range([values])

如果存在参数values，那么将这个数组形式的参数作为输出域。如果输入域是N，那么输出范围必须是N+1，如果小于N+1的元素在输出值中，那么对于任何输入比例尺将返回undefined，如果大于N+1的元素在输出值中，那么这个额外的值将被忽略，数组中的元素并不一定是数字，任何形式都可以，如果输出域未定义，那么将返回比例尺当前的输出值。

threshold.copy()

复制这个比例尺的精确副本。改变这个的比列尺不会影响返回值，反之亦然。 Log补充知识 PolyLog函数 PolyLog —普通和尼尔森（Nielsen）广义的对数函数 …… Li-*1(x)*=∫*(Li*-2(x))/x dx=∫(1-x)^(-2)+(2x)/(1-x)^3 dx=x/(1-x)^2 Li*0(x)*=∫*(Li*-1(x))/x dx=∫x/(1-x)^2 dx =x/(1-x) Li*1(x)=∫(Li_0(x))/x dx=∫1/(1-x) dx=-ln(1-x) Li_2(x)=∫(Li_1(x))/x dx=∫-(ln(1-x))/x dx Li_3(x)=∫(Li_2(x))/x dx ……*（表示下标） 对于Li*n(x) (n∈Z)*有关系式*: Li*(n+1)(x)=∫(Li_n(x))/x dx 推广到n∈R，甚至n∈C，该关系式都成立， 另外，当|x|<1 时，有关系式：(n∈C) Li_n(x) x∈C时，Li_n(x)表达式为： Li_n(x) x∈C 其中Γ(x)=(x-1)!是Gamma函数，ζ（x）是Riemann函数（zeta函数） PolyLog 函数通常用于对数积分，是数学中的特殊函数。

Linear 谭树国 20140412译 identity咕噜 20140412 00:07:00译 log 马语者20140412译 Quantize 魏飞20140412译 Threshold 20140420现明涟漪译 power Harry 20140412译 Quantile Scales Gulu 翻译 2014-11-24 23:56:33

> Wiki ▸ [[API--中文手册]] ▸ [[比例尺]] ▸ 序数比例尺

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者：
    - 邮箱：zhang_tianxu@sina.com
    - QQ群：D3数据可视化：205076374，大数据可视化：436442115
    - Github小组：VisualCrew：https://github.com/VisualCrew
- 本页译文，我们作如下约定：
    - **domain**：译为 输入域（或者定义域）
    - **range**：译为 输出范围（或值域）
    - **band**：译为 区间段（或频段宽度），可以理解为线段
    - 参数：也就是 function 的参数，译为 入参，比如：`function abc(x, y) {}`，函数 abc 的两个参数 x 和 y，我们称为：入参 x、入参 y

---

比例尺是一系列函数，用来映射输入域到输出范围。序数比例尺的输入域是离散的，比如：一组名称或类别。还有[[定量比例尺|数值比例尺]]，其输入域是连续的，比如：实数集、日期。比例尺在D3中是一个可选的功能，如果你喜欢自己处理数学的话大可不必使用它们。然而使用比例尺可以大大简化映射数据到可视化编码的必需代码量。

一个比例尺对象，例如：d3.scale.linear 的返回值既是一个对象也是一个函数。因此，可以像函数一样使用比例尺对象。并且比例尺有额外的函数可以改变它的行为。就像 D3 中的其他类，比例尺同样支持链式语法 setter 方法直接返回比例尺对象，允许多个 setter 方法在一个简洁的语句中调用。

d3.scale.**ordinal**()

构造一个新的序数比例尺，使用空的输入域和输出范围。如果序数比例尺没有指定输出范围，取值时总会返回 undefined 。

ordinal(*x*)

传入一个输入域中的值 *x*，返回对应输出范围中的值。如果输出范围已指定（比如通过 range 指定，但是不是通过 rangeBands、rangeRoundBands 或 rangePoints 来指定的），并且入参 *x* 的值不在输入域中，那么 *x* 就会被隐含地添加进到输入域中；这之后，当使用相同的值 *x* 再次调用该函数时就会返回输出范围中相同的值 *y*。

ordinal.**domain**([*values*])

获取或指定当前比例尺对象的输入域。

如果指定了入参 *values* 的值，就会设置当前比例尺对象的输入域为指定的 *values* 数组。*values* 数组中的第一个元素会被映射到输出范围中的第一个值、第二个元素会被映射到输出范围中的第二个值等等。输入域 *values* 在内部会被存储到一个关联数组中作为从值到索引的映射，生成的索引会被用来从输出范围中取值，也就是取对应索引的值；这样的话，序数比例尺的输入域中的值就会被强制转换为字符串，并且唯一地标识到输出范围中对应的值。

如果没有指定 *values*，这个方法就会返回当前的域。

序数比例尺的输入域是否指定是可选的，但是输出范围是必须明确地指定。因为，如 ordianl(x) 所述，如果 *x* 的值在输入域中不存在，就会被隐式的新增到输入域中；换句话说，输入域是从使用情况中推断而来的。虽然输入域可能被隐式构造，最好还是显式指定序数比例尺的域，以保证确定性的行为，而从使用情况来推断输入域是取决于顺序。

ordinal.**range**([*values*])

获取或指定当前比例尺对象的输出范围。

如果指定了入参 *values* 的值，就会设置当前比例尺对象的输出范围为指定的 *values* 数组。输入域中的第一个元素会被映射到 *values* 的第一个值、输入域中的第二个元素会被映射到 *values* 的第二个值等等。如果 *values* 中的值的个数少于输入域中元素的个数，那么 *values* 中的值会被循环使用。

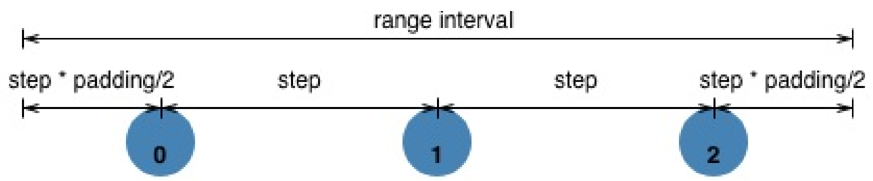
如果没有指定 *values*，这个方法就会返回当前的域。

这个函数也可以被用来指定一组已经被明确计算好了的离散的输出范围，比如：一组类目型的颜色。其他一些情况，比如：序列散点图、柱状图使用rangePoints 或 rangeBands 会更为方便，具体请先参考对应图表的案例代码。

ordinal.**rangePoints**(*interval*[, *padding*])

功能同ordinal.range([values])，只是适用范围不同。

指定输出范围为一个连续的区间 *interval* ；*interval* 需要两个数值元素，第一个表示区间的最小值、第二个表示区间的最大值。区间 *interval* 会被细分为 *n* 个等间隔的刻度“点”，*n* 的大小取决于输入域数组的真实长度（也就是数组中每个元素的唯一性而确定的长度）。在这些被细分的刻度点中，第一个点的起始位置和最后一个点的结束位置会因为入参 *padding* 的值而做相应消减（见下图），消减长度是：*padding* 个间隔长度的一半；默认情况下 *padding* 是 0 。*padding* 的值会当做间隔的倍数来使用。



```
var o = d3.scale.ordinal()
    .domain([1, 2, 3, 4])
    .rangePoints([0, 100]);

o.range(); // [0, 33.333333333333336, 66.66666666666667, 100]

o.rangePoints([0, 120], 1);
o.range()   // [15, 45, 75, 105]    前面被空了 30*1/2=15 后面也被空了 15；其中 30 是间隔宽度

o.rangePoints([0, 120], 2);
o.range()   // [24, 48, 72, 96]    前面被空了 24*2/2=24 后面也被空了 24；其中 24 是间隔宽度

o.rangePoints([0, 120], 3);
o.range()   // [30, 50, 70, 90]    前面被空了 20*3/2=30 后面也被空了 30；其中 20 是间隔宽度
```

ordinal.**rangeRoundPoints**(*interval*[, *padding*])

功能同ordinal.rangePoints(interval[, padding])，但是该函数可以美化输出的刻度点，也就是保证整数。

```
var o = d3.scale.ordinal()
    .domain([1, 2, 3, 4])
    .rangeRoundPoints([0, 100]);

o.range(); // [1, 34, 67, 100]
```

需要提及的，凑整肯定会导致额外的 *padding* 被增减，通常是和输入域的长度成一定比例；修改输出范围的区间长度，使其更紧凑便可以大大减少额外的 *padding* 被使用。

```
var o = d3.scale.ordinal()
    .domain(d3.range(50))
    .rangeRoundPoints([0, 95]);

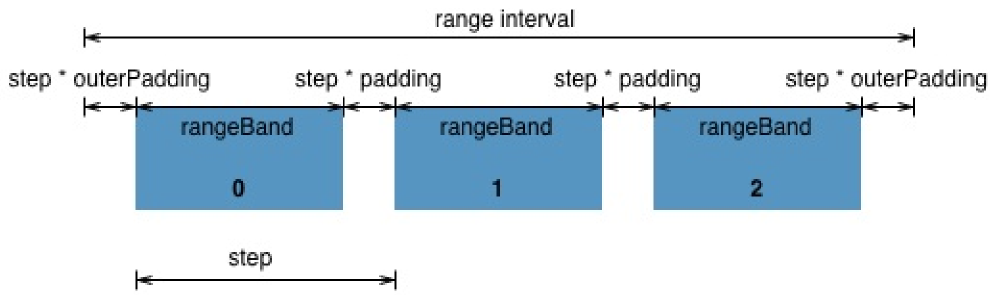
o.range(); // [23, 24, 25, …, 70, 71, 72]
o.rangeRoundPoints([0, 100]);
o.range(); // [1, 3, 5, …, 95, 97, 98]
```

（或者，你也可以手动的处理，这真的有必要吗？？？）

ordinal.**rangeBands**(*interval*[, *padding*[, *outerPadding*]])

功能同ordinal.rangePoints(interval[, padding])，但是该函数是将区间切分成一个个小的区间段，而不是一个个刻度“点”。

指定输出范围为一个连续的区间 *interval* ；*interval* 需要两个数值元素，第一个表示区间的最小值、第二个表示区间的最大值。区间 *interval* 会被切分为 *n* 个等间隔区间段，*n* 的大小取决于输入域数组的真实长度（也就是数组中每个元素的唯一性而确定的长度）。每个区间段的宽度会因为打区间的首尾 *outerPadding* 值和每个区间段的 *padding* 值而有所消减，默认情况下 *padding* 是 0 。通常，*padding* 的取值范围是 [0, 1]，表示相邻区间段间的间隔（或空白）占区间段的比例；比如： `padding=0.5` 表示区间段的实际宽度与相邻区间段间的留白相等，参考下面图片的说明。*outerPadding* 表示第一个区间的起始位置和最后一个区间的结束位置的留白，留白的长度与 *padding* 的使用方式类似， `outerPadding=0` 表示首尾顶着边缘。




```
var o = d3.scale.ordinal()
    .domain([1, 2, 3])
    .rangeBands([0, 100]);

o.rangeBand(); // 33.333333333333336
o.range(); // [0, 33.333333333333336, 66.66666666666667]
o.rangeExtent(); // [0, 100]
```

ordinal.**rangeRoundBands**(*interval*[, *padding*[, *outerPadding*]])

功能同rangeBands，但是该函数可以美化输出的区间段，也就是保证每个区间段的起点值都是整数。

```
var o = d3.scale.ordinal()
    .domain([1, 2, 3])
    .rangeRoundBands([0, 100]);

o.range(); // [1, 34, 67]
o.rangeBand(); // 33
o.rangeExtent(); // [0, 100]
```

需要提及的，凑整肯定会导致额外的 *padding* 被增减，通常是和输入域的长度成一定比例；修改输出范围的区间长度，使其更紧凑便可以大大减少额外的 *padding* 被使用。

```
var o = d3.scale.ordinal()
    .domain(d3.range(50))
    .rangeRoundBands([0, 95]);

o.range(); // [23, 24, 25, …, 70, 71, 72]

o.rangeRoundBands([0, 100]);
o.range(); // [0, 2, 4, …, 94, 96, 98]
```

（或者，你也可以手动的处理，这真的有必要吗？？？）

ordinal.**rangeBand**()

获取区间段的宽度。只有当使用 rangeBands 或 rangeBands 来指定输出范围时才有效，否则返回一律返回 0。

ordinal.**rangeExtent**()

获取当前比例尺对象的未被切分的输出范围：一个两元素的数组，第一个元素表示最小值、第二个元素表示最大值。

ordinal.**copy**()

深度拷贝一个当前比例尺对象的副本，并返回，更改这个副本的属性不会影响到原比例尺对象的属性。

Categorical Colors

d3.scale.**category10**()

构造一个新的序数比例尺，使用以下10种类型的颜色：

#1f77b4
#ff7f0e
#2ca02c
#d62728
#9467bd
#8c564b
#e377c2
#7f7f7f
#bcbd22
#17becf

d3.scale.**category20**()

构造一个新的序数比例尺，使用以下20种类型的颜色：

#1f77b4
#aec7e8
#ff7f0e
#ffbb78
#2ca02c
#98df8a
#d62728
#ff9896
#9467bd
#c5b0d5
#8c564b
#c49c94

- #e377c2
- #f7b6d2
- #7f7f7f
- #c7c7c7
- #bcbd22
- #dbdb8d
- #17becf
- #9edae5

d3.scale.**category20b**()

构造一个新的序数比例尺，使用以下20种类型的颜色：

- #393b79
- #5254a3
- #6b6ecf
- #9c9ede
- #637939
- #8ca252
- #b5cf6b
- #cedb9c
- #8c6d31
- #bd9e39
- #e7ba52
- #e7cb94
- #843c39
- #ad494a
- #d6616b
- #e7969c
- #7b4173
- #a55194
- #ce6dbd
- #de9ed6

d3.scale.**category20c**()

构造一个新的序数比例尺，使用以下20种类型的颜色：

- #3182bd
- #6baed6
- #9ecae1

#c6dbef
#e6550d
#fd8d3c
#fdae6b
#fdd0a2
#31a354
#74c476
#a1d99b
#c7e9c0
#756bb1
#9e9ac8
#bcbddc
#dadaeb
#636363
#969696
#bdbdbd
#d9d9d9

ColorBrewer

D3也通过 [[Cynthia Brewer|http://colorbrewer2.org/]] 绑定了一些奇妙的类目颜色比例尺。可以从 lib/colorbrewer 找到这些颜色比例尺的 CSS 或 Javascript 实现。

对于 CSS，只要在需要着色的元素上指定 class 类如：像"q0-3"、"q1-3"或"q2-3"，然后，在父元素（例如SVG元素）上设置需要的颜色比例尺的名称到 class 类属性如"RdBu"或"Blues"。具体参见： calendar heatmap、choropleth

对于 JavaScript，可以使用 colorbrewer.RdBu[9] 或等同的方法作为 **d3.scale.ordinal** 的范围如：

```
var o = d3.scale.ordinal()
    .domain(["foo", "bar", "baz"])
    .range(colorbrewer.RdBu[9]);
```

Gulu@VisualCrew小组 译于 2014-11-24 23:21:05

WeiFei365@VisualCrew小组 校验于 2015-12-01 22:32:09

- 目前，本页结构完全和英文页相同
- 已同步更新首页的链接；

- 其他页面的引用可以像英文文档那样直接引用；

> Wiki ▸ [[API--中文手册]] ▸ [[SVG函数]]

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

参见：

- [[形状|SVG-形状]] - 生成SVG形状。
- [[轴|SVG-轴]] - 生成SVG标准轴。
- [[控件|SVG-控制]] - 生成SVG标准控件。

> Wiki ▸ [[API--中文手册]] ▸ [[SVG函数]] ▸ **SVG 形状**

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

SVG有大量的内置建议图形。例如：轴对齐矩形和圆形。为了更大的灵活性，可以使用SVG的路径（path）元素结合D3的路径数据生成器。如果你熟悉Protovis，你会注意到，D3的路径生成类似Protovis的标记（marks）。 形状生成器，例如d3.svg.arc返回的，既是一个对象又是一个函数。也就是说，你可以像其他函数一样调用shape，并且shape含有额外的方法来改变它的行为。像D3中的其他类一样，shape遵循链式语法由setter方法返回shape自己。允许在一个简单的申明中调用多个setter方法。 SVG Elements

所有的SVG形状都可以使用transform属性转换。你可以将转换直接应用在shape上，或者应用在含g的元素上。这样，当形状是定义为“轴对齐”的，这只意味着在本地坐标系中轴对齐；你可以旋转或者转换形状。形状可使用fill 和 stroke样式填充或者描边（你也可以使用同名称的属性，但是当兼容外部样式时推荐使用样式）。

svg:rect x="0" y="0" width="0" height="0" rx="0" ry="0"

Rect矩形元素定义一个轴对齐的矩形。矩形的左上角使用x，y属性定位，使用width和height指定尺寸。圆角矩形可以使用可选参数rx 和 ry生成。

svg:circle cx="0" cy="0" r="0"

circle圆形元素定义了一个基于中心点和半径的圆形。使用cx和cy属性指定圆的中心。使用r属性指定圆的半径。

svg:ellipse cx="0" cy="0" rx="0" ry="0"

Ellipse椭圆元素定义了一个轴对齐的椭圆，基于一个中心点和两个半径。中心点使用cx和cy属性定位。而半径使用rx 和ry指定。

svg:line x1="0" y1="0" x2="0" y2="0"

Line直线元素定义线段始于一个点而终止于另一个点。第一个点使用x1和y1属性指定，第二个点使用x2和y2属性指定。线元素常用来画规则，参考线，轴和刻度线。

svg:polyline points=""

polyline 折线元素定义一组相连的直线段。通常，polyline 元素定义开放形状。使用points属性指定构成折线的点。注意：在D3中通常将d3.svg.line路径生成器和path元素一起使用会更方便灵活。

svg:polygon points=""

polygon 多边形元素定义一个由一组相连的直线段构成的闭合图形。使用points属性指定构成多边形的点。注意：在D3中通常将d3.svg.line路径生成器和path元素一起使用会更方便灵活。线可以使用路径关闭命令"Z"闭合。

svg:text x="0" y="0" dx="0" dy="0" text-anchor="start"

Text文本元素定义文本组成的图形元素。文本元素的文本内容（参见文本操作符）定义用来渲染的字符。文本元素的锚点使用x，y属性控制。另外，文本可以使用dx，dy属性偏移。这个偏移在你可以使用与字体大小相关的“em”单元控制文本的边沿和基线时尤其方便。横向文本对齐由text-anchor属性控制，下面是一些例子：

left-align, bottom-baseline</svg:text>

center-align, bottom-baseline</svg:text>

right-align, bottom-baseline</svg:text>

left-align, middle-baseline</svg:text>

center-align, middle-baseline</svg:text>

right-align, middle-baseline</svg:text>

left-align, top-baseline</svg:text>

center-align, top-baseline</svg:text>

right-align, top-baseline</svg:text> 有可能有一个更好的方法来指定文字的基线，使用SVG的基线对齐属性，但是这不是被浏览器广泛地支持的。最后，字体的颜色通常使用fill样式指定（你也可以使用stroke），字体使用font，font-family，font-size和相关的样式控制。一些浏览

器也支持CSS3属性，例如text-shadow。

svg:path d="" transform=""

Path路径元素代表了形状的轮廓可以被填充，描边，用做剪裁路径，或者这三者的任意组合。d属性定义了路径数据，这是路径命令的一个迷你语言，例如moveto (M)，lineto (L)和closepath (Z)。路径元素是一个SVG中所有其他形状的概括，几乎可以用来画出任何东西！Path Data Generators

为了简化路径元素的d属性的构造，D3包含很多辅助类用来生成路径数据。如果你熟悉Protovis，你就会发现这些路径生成器和Protovis的标记类型是相似的：每个生成器就是一个数据的函数。所以，如果你的数据是xy坐标序列。你可以定义访问器函数，路径生成器用来制造路径数据。例如，你可以这样定义生成器： var line = d3.svg.line() .x(function(d) { return d.x; }) .y(function(d) { return d.y; }) .interpolate("basis"); 然后，你可以使用下面函数设置d属性： g.append("path") .attr("d", line); 不论数据是否绑定到g元素（在这个例子中）都将传递给line实例。这样，数据必须指定为一个数组。对于数据数组中的每个元素，x和y访问器函数用来抽出控制点坐标。 路径生成器，例如d3.svg.line返回的既是一个函数又是一个对象。这样，你就可以像其他函数一样调用生成器了。并且生成器还有额外的方法来改变它的行为。像D3中的其他类一样，路径生成器遵循方法链模式其中的setter方法返回生成器自身。允许在一个简单的声明中调用多个setter方法。

d3.svg.line()

构造一个新的线生成器使用默认的x和y访问器函数（假设输入数据是一个两元素数字数组；详见下文），和线性插值器。返回的函数生成路径数组为开口分段线性曲线，折线或者，如折线图：

通过改变插值器，你可以生成样条线以及步长函数。另外，不要害怕最后粘上其他路径命令。例如，如果你想生成一个封闭的路径，追加closepath (Z)命令： g.append("path") .attr("d", function(d) { return line(d) + "Z"; });

线生成器设计来和面积生成器一起使用。例如，当生成面积图时，你可能使用带有填充属性的面积生成器，以及带有描边属性的线生成器来突出面积的上边缘。当线生成器只是用来设置d属性，你可以使用SVG样式和属性控制线的展示，例如fill，stroke和stroke-width。

line(data)

为指定的data元素数组返回路径数据字符串，如果路径是空则返回null。

line.x([x])

如果指定了x，为指定的函数或常量设置x访问器。如果x没有指定，返回当前的x访问器。这个访问器为传递给x线生成器的数据数组中的每个元素。默认的访问器假设每个输入的元素是一个二元素数字数组： function x(d) { return d[0]; }

通常，一个x访问器是指定的，因为输入数据是不同格式的，或者因为你想使用比例尺。例如，如果你的数据指定为一个含有x，y属性的的对象，而不是一个元组，你可以反引用这些属性同时应用比例尺： var x = d3.scale.linear().range([0, w]), y = d3.scale.linear().range([h, 0]);

var line = d3.svg.line() .x(function(d) { return x(d.x); }) .y(function(d) { return y(d.y); });

X访问器像D3中其他值函数一样的方式调用。函数的This上下文就是选择中的当前元素（通常，相同的this上下文调用线函数；然而，在通常情况下线生成器传递给attr操作符，this上下文将关联DOM元素）。函数传入两个入参当前的数据d和当前的索引i。this上下文，索引就是控制点中的索引，而不是选择中当前元素的索引。X访问器按照数据数组中指定的顺序，每个元素恰好调用一次。这样，就可能指定不确定性访问器，例如随机数生成器。也可以指定x访问器为一个常量而不是函数，在所有点都有相同的x坐标情况下。

line.y([y])

如果指定了y，为指定的函数或常量设置y访问器。如果y没有指定，返回当前的y访问器。这个访问器为传递给y线生成器的数据数组中的每个元素。默认的访问器假设每个输入的元素是一个二元素数字数组： function y(d) { return d[1]; } 怎样指定一个y访问器，参见相似的x访问器。 注意：像所有的其他图形库，SVG使用左上角作为原点。这样在屏幕中较高的y值就更低。对于可视化我们常常想要原点是在左下角。一个简单的方法实现这个可以使用range([h, 0])替代range([0, h])反转y比例尺的范围。

line.interpolate([interpolate])

如果指定了interpolate 参数，就会设置插值器模式为指定的字符串或者函数。如果没有指定，就返回当前的插值器模式。支持下面的命名插值器模式：

• linear -分段线性片段，如折线。 • linear-closed –闭合直线段，以形成一个多边形。 • step -水平和垂直段之间交替，如在step函数中。 • step-before -垂直和水平段之间交替，如在step函数中。 • step-after -水平和垂直段之间交替，如在step函数中。 • basis - B样条曲线，在两端的控制点的重复。 • basis-open – 开放B样条曲线，首尾不会相交。 • basis-closed -封闭B样条曲线，如在一个循环。 • bundle – 等价于basis, 除了使用tension参数拉直样条曲线。 •

cardinal – 基本样条曲线，在末端控制点的重复。 • cardinal-open –开放的基本样条曲线，首尾不会相交，但和其他控制点相交。 • cardinal-closed -封闭基本样条曲线，如在一个循环。 • monotone - 三次插值，可以保留y的单调性。

其中一些插值模式的行为可能通过指定的张力tension进行进一步定制。 如果interpolate 是一个函数，然后这个函数将被调用来反转一个形如[[x0, y0], [x1, y1], …]的点数组，返回一个SVG路径数据字符串，用来展示线。字符串开始处的“M” 隐含的，不应该被返回。例如，线性插值被实现为 ： function interpolateLinear(points) { return points.join("L"); } 这相当于（并且比这更有效） ： function interpolateLinear(points) { var path = ""; for (var i = 0; i < points.length; i++) { if (i) path += "L"; path += points[i][0] + "," + points[i][1]; } return path; } 参见bl.ocks.org/3310323是另外一个自定义线性插值器。

line.tension([tension])

如果指定tension 参数，设置基本样条曲线插值器拉伸为指定的范围[0, 1]内的数字。如果没有指定tension 放回当前的拉伸。拉伸只在基本样条曲线模块有效：样条曲线、开放样条曲线和闭合样条曲线。默认拉伸值是0.7。在某种意义上，这可以解释为切线的长度； 1将产生全为零的切线，0产生Catmull-Rom样条曲线。

注意：拉伸必须指定为一个常量，而不是函数，因为它是整个线的常数。但是，也可能使用同一个生成器生成的不同拉伸生成多条线。例如： svg.selectAll("path") .data([0, 0.2, 0.4, 0.6, 0.8, 1]) .enter().append("path") .attr("d", function(d) { return line.tension(d)(data); });

在这个例子中（见live version），拉伸是在线生成器调用之前设置的，这样就可以在线中使用相同的数据生成不同的路径了。 live version：http://bl.ocks.org/mbostock/1016220

line.defined([defined])

取得或者设置访问器函数控制线的生成。如果指定了defined 参数，就设置新的访问器函数并返回线。如果没有指定defined ，就返回当前生成器默认就是function() { return true; }。定义的访问器可以用来定义是否定义线，通常在和缺失数据一起使用时很有用。生成的数据将自动被分成多个不同的子路径，跳过未定义数据，例如，如果你想忽略非数字（或者未定）的y值，可以这样写： line.defined(function(d) { return !isNaN(d[1]); });

d3.svg.line.radial()

构造一个新的径向线生成器使用默认的半径和角度访问器函数（假设输入数据是一个两元素的数字数组；详见下文），和一个线性插值器。返回的函数为开放的分段线性曲线，或折线与笛卡尔线生成器生成路径数据。

line(data)

为指定的data元素数组返回路径数据字符串

line.radius([radius])

function radius(d) { return d[0]; } 如果指定了radius 参数，则设置半径访问器为指定的函数或常量。如果radius 没有指定，返回当前的半径访问器。这个访问器为传递给线生成器的数据数组中的每个元素调用。默认的访问器假设每个输入元素都是一个两元素的数字数组：function radius(d) { return d[0]; } 这个方法是笛卡尔line.x方法的变形。

line.angle([angle])

如果指定了angle 参数，设置角度访问器为指定的函数或者弧度为单位的常量。如果没有指定angle 参数，返回角度访问器。访问器为传递给线生成器的数据数组中的每个元素调用。默认的访问器假设每个输入元素是一个两元素的数字数组。 function angle(d) { return d[1]; } 如果指定了angle 参数，设置角度访问器为指定的函数或者弧度为单位的常量。如果没有指定angle 参数，返回角度访问器。访问器为传递给线生成器的数据数组中的每个元素调用。默认的访问器假设每个输入元素是一个两元素的数字数组。 function angle(d) { return d[1]; } 这个方法就是笛卡尔line.y 方法的变形。

line.interpolate([interpolate])

参见笛卡尔 line.defined方法， 投影到笛卡尔空间之后惨发生插值。

line.tension([tension])

参见笛卡尔 line.tension方法， 投影到笛卡尔空间之后惨发生插值。

line.defined([defined])

参见笛卡尔 line.defined方法

d3.svg.area()

构造一个新的面积生成器，使用默认的x（横坐标），y0（基线），y1（顶线）访问器函数（假定输入数据是一个双元素数字数组，详见下文）和线性插值器。对于分段线性曲线，或多边形，返回的函数生成一个封闭的path数据，如在面积图中:

从概念上讲，多边形的形成是通过使用两条线：顶线的形成是通过x（横坐标）和y1（顶线）访问器方法，从左至右生成。底线是添加到这一条线上，使用x （横坐标）和y0（基线）访问器方法，从右到左去生成。通过将转换属性设置为旋转path元素90度，您还可以生成垂直的面积图。通过改变插值，您还可以生成splines和台step函数。

区域生成器被设计为与线生成器一起使用的。例如，当生成区域图表时，你可能用一个带有填充样式的区域生成器和一个带有描边样式的线生成器，此描边样式可以使区域的顶部边缘更加突出显示。由于区域生成器只可以用于设置d属性，所以你可以通过使用标准的SVG样式和属性去控制区域的外观，例如填充样式。

创建streamgraphs (堆叠区域图)，使用stack（叠层）布局。此布局可以为一个系列中的每个值去设置y0属性，这个系列值可从y0（基线），y1（顶线）访问器中使用。注意，每个系列中必须有相同数量的值，并且每个值必须有相同的x横坐标；如果你在每系列里有缺失数据或不一致的x横坐标，那么在计算叠层布局之前，你必须重新取样和插入你的数据。

area(data)

对于指定的数据元素数组，返回路径（path）数据字符串；当路径是空的时候返回null。

area.x([x])

如果指定了x参数，设置x（横坐标）访问器为指定的方法或常量。如果没有指定x，返回当前x访问器。为传递给区域生成器的数据数组中的每个元素，调用该访问器。默认访问器中假定每个输入元素是双元素数字数组:

function x(d) { return d[0]; }

通常情况下，x访问器被指定是因为输入数据是不同格式的，或者因为你想应用一个比例尺。例如，如果你的数据被指定为一个带有x和y属性的对象，而不是一个元组，那么你可能会引用这些属性同时应用比例尺：

var x = d3.scale.linear().range([0, w]), y = d3.scale.linear().range([h, 0]);

var area = d3.svg.area() .x(function(d) { return x(d.x); }) .y0(h) .y1(function(d) { return y(d.y); });

X访问器会和D3中其他值函数一样的方式被调用。函数中的this上下文就是选择中的当前元素。（从技术上讲，相同的this上下文调用区域函数；然而，通常情况下，区域生成器被传递到attr操作符，this上下文将会被关联到DOM元素上）。函数被传递两个参数，当前的数据

（d）和当前的索引值（i）。在this上下文中，索引值就是控制数据点的数组中的索引，而不是当前选择元素的索引。x访问器在每一个数据中按照数据数组指定的顺序被恰好调用一次。因此，可以指定一个不确定的访问器，例如随机数生成器。也可以指定x访问器为一个常量，而不是一个函数，在这种情况下，所有的点将有相同的横坐标值。

area.x0([x0])

…

area.x1([x1])

…

area.y([y])

…

area.y0([y0])

如果y0被指定，y0（基线）访问器为指定的方法或常量。如果没有指定y0，返回当前y0（基线）访问器。将为传递给区域生成器数据数组中的每个元素，调用该访问器函数。默认的访问器是常量0，也就是使用一个固定的基线y = 0。对于如何指定一个y0（基线）访问器的例子，请看类似的x访问器。

area.y1([y1])

如果指定参数y1，y1访问器为指定的方法或常量。如果没有指定y1，就会返回当前的y1访问器。将为传递给区域生成器数据数组中的每个元素，调用该访问器函数。默认的访问器假定每个输入元素是一个双元素的数字数组: function y1(d) { return d[1]; }

关于如何指定一个y1（顶线）访问器的一个例子，请看类似的x访问器。注意，像大多数其他的图形库，SVG使用顶部-左侧作为原点，因此更高的数值y将会在屏幕更低的位置。对于可视化而言，我们通常希望原点是位于底部-左侧的位置。可以做到这一点的一个简单的方法便是转化y比例尺的范围，即通过使用范围([h,0])而不是范围([0,h])。

area.interpolate([interpolate])

如果指定插值器，便设置插值器模式为指定的字符串或函数。如果没有指定插值器，返回当前插值器模式。插值器模式支持以下命名： linear（线性）：分段的线性片段，如折线。 step（步）：水平和垂直片段之间交替，如台阶函数。 step-before：垂直和水平片段之间交替，如台阶函数。 step-after： 水平和垂直片段之间交替，如台阶函数。 basis ：一个B-spline，在末尾控制点的重复。 basis-open ：一个开放的B-spline；首尾不相交。 cardinal ：一个Cardinal spline，在末尾控制点的重复。 cardinal-open：一个开放的Cardinal spline；首尾不相交，但是会和其他控制点相交。 monotone -立方插值保存y值得单调性。

其中一些插值模式的行为通过指定的张力（tension）可能被进一步自定义。从技术上讲,同样也会支持basis-closed和 cardinal-closed的插值模式，但这些模式应用在一条线上比应用在一个区域上更有意义。 如果参数interpolate是一个函数，那么这个函数将被调用去转换一个形式为[[x0, y0], [x1, y1], …]的数组，返回一个SVG路径数据字符串，它将用于显示区域。字符串起始位置的“M”是隐含的，不会被返回。例如，线性插值的实现为: function interpolateLinear(points) { return points.join("L"); } This is equivalent to (and more efficient than): function interpolateLinear(points) { var path = ""; for (var i = 0; i < points.length; i++) { if (i) path += "L"; path += points[i][0] + "," + points[i][1]; } return path; } See bl.ocks.org/3310323 for another example of custom interpolation. 这个（比下边效率更高）相当于：functioninterpolateLinear(points){ varpath=""; for(vari=0;i<points.length;i++){ if(i)path+="L"; path+=points[i][0]+","+points[i][1]; } returnpath; } 请查看另一个自定义插值的样例：bl.ocks.org/3310323

area.tension([tension])

如果指定tension，便将基数样条(Cardinal spline)插值张力（tension）设置为指定区间[0,1]内的数字。如果没有指定张力，返回当前的张力值。张力值只会影响基本插值模式：cardinal, cardinal-open 和cardinal-closed。默认的张力值是0.7。在某种意义上，这可以解释为切线长度；值1将产生零切线，值0将产生一个Catmull-Rom spline。注意，张力值必须指定为一个常数，而不是一个函数，因为它是整个区域的常数。

area.defined([defined])

获取或设置访问器函数，此方法用来控制已经明确定义的区域。如果指定参数defined（定义），便设置新的取值函数（访问器）并返回该区域。如果没有指定defined（定义），则返回当前访问器的默认函数，即function(){return true;}。定义访问器可用于定义哪些区域是定义的，哪些是未定义的，通常对于有数据缺失情况是很有用的；生成的路径数据将自动被分为多个不同的子路径，同时跳过未定义的数据。例如，如果您想要忽略那些不是一个数字的（或未定义的）y值，你可以这样写: area.defined(function(d) { return !isNaN(d[1]); });

d3.svg.area.radial()

…

area(data)

对于指定的数组中的数据元素（data），返回路径数据字符串。

area.radius([radius])

…

area.innerRadius([radius])

…

area.innerRadius([radius])

…

area.angle([angle])

…

area.startAngle([angle])

…

area.endAngle([angle])

…

d3.svg.arc()

构造一个新的弧生成器，使用默认的内半径、外半径、开始弧度和结束弧度访问器（假定输入数据是包含匹配于访问器的命名属性的对象，详见下文）；而默认的访问器假定弧的尺寸是动态指定的，当然设置一个或多个的尺寸为常量也是很常见的，例如饼图的内半径设为0；返回的函数为封闭的实心弧度生成路径数据，如饼图或环图：

事实上，有四种可能性：圆circle（内半径为0，角度跨度大于等于2π），扇形circular sector（内半径为0，角度跨度小于2π），环形 annulus（内半径大于0，角度跨度为2π），以及环形扇区（内半径大于0，并且角度跨度小于2π）。

arc(datum[, index])

为指定的datum参数返回路径数据字符串。可选参数index 可能会指定，传递给弧度访问器函数。

arc.innerRadius([radius])

如果指定radius，则设置内半径访问器outerRadius-accessor为指定值或函数；如果未指定，则返回当前的内半径访问器；访问器参数传递给弧生成器时调用；默认访问器假定输入数据是带有适当命名属性的对象：

function innerRadius(d) { return d.innerRadius; }

通常情况下，指定内半径访问器 innerRadius-accessor是由于：输入数据是不同的格式、或你想要应用比例尺，亦或你想为圆环指定个恒定的内半径。 内半径访问器会像D3的其他函数一样被调用；函数中的this代表选择中的当前元素（技术上来说，this上下文调用弧函数；然而一般情况下，弧生成器传递给attr操作符， this上下文将关联DOM元素）；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数而不是函数。

arc.outerRadius([radius])

如果指定radius，则设置外半径访问器outerRadius-accessor为指定值或函数；如果未指定，则返回当前的外半径访问器；访问器参数传递给弧生成器时调用；默认访问器假定输入数据是带有适当命名属性的对象：

function outerRadius(d) { return d.outerRadius; }

通常情况下，指定外半径访问器 innerRadius-accessor是由于：输入数据是不同的格式、或你想要应用比例尺，亦或你想为圆环指定个恒定的外半径。 外半径访问器会像D3的其他函数一样被调用；函数中的this代表选择中的当前元素（技术上来说，this上下文调用弧函数；然

而一般情况下，弧生成器传递给attr操作符， this上下文将关联DOM元素）；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数而不是函数。

arc.startAngle([angle])

如果指定angle，则设置开始角度访问器startAngle-accessor为指定的函数或常数，如果未指定，则返回当前的访问器；角度使用弧度radians表示，尽管SVG中使用的是角度；角度为0对应12点钟的指针方向（负数y）并且顺时钟方向继续旋转2π；访问器参数传递给弧生成器时调用；默认访问器假定输入数据是带有适当命名属性的对象：

function startAngle(d) { return d.startAngle; }

为了构建饼图和环图，需要计算每个弧的起始角度和上个弧的结束角度；使用饼布局 pie会非常方便的，即类似于堆叠布局stack；给定一组输入数据，饼布局会调用弧对象来生成开始弧度和结束弧度属性，你也可以使用默认的弧访问器。 开始角度访问器startAngle-accessor会像D3的其他函数一样被调用；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数。

arc.endAngle([angle])

如果指定angle，则设置开始角度访问器endAngle-accessor为指定的函数或常数，如果未指定，则返回当前的访问器；角度使用弧度radians表示，尽管SVG中使用的是角度；访问器会以传递给弧度生成器的参数形式被调用；默认访问器假定输入数据是包含适当命名属性的对象： function endAngle(d) { return d.endAngle; } 为了构建饼图和环图，需要计算每个弧的起始角度和上个弧的结束角度；这保证了你会非常方便的使用饼布局 pie，即类似于堆叠布局stack；给定一组数据，饼布局会调用弧对象来生成开始弧度和结束弧度属性，你也可以使用默认的弧访问器。 开始角度访问器endAngle-accessor会像D3的其他函数一样被调用；函数传递两个参数：当前数据d和索引i；当然，也可以指定其为一个常数。

arc.centroid(arguments…)

计算以指定输入参数产生的弧的圆心，通常情况下，参数为：当前数据d和索引i；圆心被定义为内外半径和开始结束角度的极坐标系的中心点；这为圆弧的标签提供了方便的位置信息，例如： arcs.append("text") .attr("transform", function(d) { return "translate(" + arc.centroid(d) + ")"; }) .attr("dy", ".35em") .attr("text-anchor", "middle") .text(function(d) { return d.value; }); 或者，你可以使用SVG的变换transform 属性来旋转文本的位置，即使你可能需要转换弧度为角度；另一种可能性是使用 textPath元素来依照弧的path路径来弯曲文本标签显示。

d3.svg.symbol()

Constructs a new symbol generator with the default type- and size-accessor functions (that make no assumptions about input data, and produce a circle sized 64 square pixels; see below for details). While the default accessors generate static symbols, it is common to set one or more of the accessors using a function, such as setting the size proportional to a dimension of data for a scatterplot. The returned function generates path data for various symbols, as in a dot plot:

Note that the symbol does not include accessors for x and y. Instead, you can use the path element's transform attribute to position the symbols, as in: vis.selectAll("path") .data(data) .enter().append("path") .attr("transform", function(d) { return "translate(" + x(d.x) + "," + y(d.y) + ")"; }) .attr("d", d3.svg.symbol()); In the future, we may add x- and y-accessors for parity with the line and area generators. The symbol will be centered at the origin (0,0) of the local coordinate system. You can also use SVG's built-in basic shapes to produce many of these symbol types, though D3's symbol generator is useful in conjunction with path elements because you can easily change the symbol type and size as a function of data.

symbol(datum[, index])

Returns the path data string for the specified datum. An optional index may be specified, which is passed through to the symbol's accessor functions.

symbol.type([type])

If type is specified, sets the type-accessor to the specified function or constant. If type is not specified, returns the current type-accessor. The default accessor is the constant "circle", and the following types are supported: • circle - a circle. • cross - a Greek cross or plus sign. • diamond - a rhombus. • square - an axis-aligned square. • triangle-down - a downward-pointing equilateral triangle. • triangle-up - an upward-pointing equilateral triangle. Types are normalized to have the same area in square pixels, according to the specifiedsize. However, note that different types' sizes may be affected by the stroke and stroke width in different ways. All of the types are designed to be visible when only a fill style is used (unlike the Protovis cross), although they generally look better when both a fill and stroke is used. The type-accessor is invoked in the same manner as other value functions in D3. The thiscontext of the function is the current element in the selection. (Technically, the same thiscontext that invokes the arc function; however, in the common case that the symbol generator is passed

to the attr operator, the this context will be the associated DOM element.) The function is passed two arguments, the current datum (d) and the current index (i). It is also possible to specify the type-accessor as a constant rather than a function.

symbol.size([size])

If size is specified, sets the size-accessor to the specified function or constant in square pixels. If size is not specified, returns the current size-accessor. The default is 64. This accessor is invoked on the argument passed to the symbol generator. Typically, a size-accessor is specified as a function when you want the size of the symbol to encode aquantitative dimension of data, or a constant it you simply want to make all the dots bigger or smaller. If you want to specify a radius rather than the size, you must do so indirectly, for example using a pow scale with exponent 2.

d3.svg.symbolTypes

The array of supported symbol types.

d3.svg.chord()

Constructs a new chord generator with the default accessor functions (that assume the input data is an object with named attributes matching the accessors; see below for details). While the default accessors assume that the chord dimensions are all specified dynamically, it is very common to set one or more of the dimensions as a constant, such as the radius. The returned function generates path data for a closed shape connecting two arcs with quadratic Bézier curves, as in a chord diagram:

A chord generator is often used in conjunction with an arc generator, so as to draw annular segments at the start and end of the chords. In addition, the chord layout is useful for generating objects that describe a set of grouped chords from a matrix, compatible with the default accessors.

chord(datum[, index])

Returns the path data string for the specified datum. An optional index may be specified, which is passed through to the chord's accessor functions.

chord.source([source])

If source is specified, sets the source-accessor to the specified function or constant. If sourceis not specified, returns the current source-accessor. The purpose of the source accessor is to return an object that describes the starting arc of the chord. The returned object is subsequently passed to the radius, startAngle and endAngle accessors. This allows these other accessors to be reused for both the source and target arc descriptions. The default accessor assumes that the input data is an object with suitably-named attributes: function source(d) { return d.source; } The source-accessor is invoked in the same manner as other value functions in D3. The thiscontext of the function is the current element in the selection. (Technically, the same thiscontext that invokes the arc function; however, in the common case that the symbol generator is passed to the attr operator, the this context will be the associated DOM element.) The function is passed two arguments, the current datum (d) and the current index (i). It is also possible to specify the source-accessor as a constant rather than a function.

chord.target([target])

If target is specified, sets the target-accessor to the specified function or constant. If target is not specified, returns the current target-accessor. The purpose of the target accessor is to return an object that describes the ending arc of the chord. The returned object is subsequently passed to the radius, startAngle and endAngle accessors. This allows these other accessors to be reused for both the source and target arc descriptions. The default accessor assumes that the input data is an object with suitably-named attributes: function target(d) { return d.target; } The target-accessor is invoked in the same manner as other value functions in D3. The function is passed two arguments, the current datum (d) and the current index (i). It is also possible to specify the target-accessor as a constant rather than a function.

chord.radius([radius])

If radius is specified, sets the radius-accessor to the specified function or constant. If radiusis not specified, returns the current radius-accessor. The default accessor assumes that the input source or target description is an object with suitably-named attributes: function radius(d) { return d.radius; } The radius-accessor is invoked in a similar manner as other value functions in D3. The function is passed two arguments, the current source description (derived from the current datum, d) and the current index (i). It is also possible to specify the radius-accessor as a constant rather than a function.

chord.startAngle([angle])

If startAngle is specified, sets the startAngle-accessor to the specified function or constant. IfstartAngle is not specified, returns the current startAngle-accessor. Angles are specified inradians, even though SVG typically uses degrees. The default accessor assumes that the input source or target description is an object with suitably-named attributes: function startAngle(d) { return d.startAngle; } The startAngle-accessor is invoked in a similar manner as other value functions in D3. The function is passed two arguments, the current source or target description (derived from the current datum, d) and the current index (i). It is also possible to specify the startAngle-accessor as a constant rather than a function.

chord.endAngle([angle])

If endAngle is specified, sets the endAngle-accessor to the specified function or constant. IfendAngle is not specified, returns the current endAngle-accessor. Angles are specified inradians, even though SVG typically uses degrees. The default accessor assumes that the input source or target description is an object with suitably-named attributes: function endAngle(d) { return d.endAngle; } The endAngle-accessor is invoked in a similar manner as other value functions in D3. The function is passed two arguments, the current source or target description (derived from the current datum, d) and the current index (i). It is also possible to specify the endAngle-accessor as a constant rather than a function.

d3.svg.diagonal()

使用默认的访问函数（即假设输入数据是一个属性名与访问器相匹配的对象；详见下文）构造一个新的对角线生成器；返回的函数会生成一条贝塞尔曲线（path）来连接source和target两端；当连接节点时，切线被指定用来产生平滑的扇入扇出，如一个node-link diagram图：

虽然对角线默认为笛卡尔（轴对齐）的方向，但是可以用于使用投影的径向或其他方向；

diagonal(datum[, index])

根据指定的数据datum返回path的路径数据字符串；一个可选的index 属性可以被指定，被用来传递给对角线生成器的函数。

diagonal.source([source])

如果source 指定，设置source 访问器为指定的函数或常量；如果source 未指定，返回当前的source 访问器；source 访问器旨在返回描述对角线起点的对象，返回的对象随后将会被传递给projection；默认的访问器是假定输入数据对象的属性名和访问器相匹配的，如： function source(d) { return d.source; }

source 访问器就像D3中其他值函数一样方式被调用的；函数的this 就是当前元素的选择；（技术上来说，调用对角线函数的this 是相同的；然而，通常情况下，这个符号生成器被传递给了attr 操作符，this 上下文将关联DOM元素）；函数需要传递两个参数：当前的数据d和当前所在索引i；另外，也可以指定source 访问器为一个常数；

diagonal.target([target])

如果target 指定，设置target 访问器为指定的函数或常量；如果target 未指定，返回当前的target 访问器；target 访问器旨在返回描述对角线终点的对象，返回的对象随后将会被传递给projection；默认的访问器是假定输入数据对象的属性名和访问器相匹配的，如： function target(d) { return d.target; }

target 访问器就像D3中其他值函数一样方式被调用的；函数的this 就是选择中的当前元素；函数需要传递两个参数：当前的数据d和当前所在索引i；另外，也可以指定target 访问器为一个常数而不是函数。

diagonal.projection([projection])

如果投影projection 指定，设置projection 为指定的函数；如果projection 未指定，返回当前的projection ；投影projection转换形如的点{x, y}（例如由来源和目标返回的）为两元素的数字数组；默认的访问器假定输入端是一个包含x 和y 两个属性的对象，如： function projection(d) { return [d.x, d.y]; }

默认的访问器是兼容D3的布局的，包括：tree、cluster和partition；例如，要产生一个径向对角线，假定属性y是以像素为单位，并且x属性是定义以度为单位的角度，像这样： function projection(d) { var r = d.y, a = (d.x - 90) / 180 *Math.PI; return [r* Math.cos(a), r * Math.sin(a)]; }

projection 是类似于D3中其他值函数一样方式被调用的；这个projection 函数需要传递两个参数：当前的source 或target 端（从当前数据d中得到）和当前的索引i；

d3.svg.diagonal.radial()

…

diagonal(datum[, index])

根据指定的数据datum返回path的路径数据字符串；一个可选的index 属性可以被指定，被用来传递给对角线生成器的函数。

- SVG元素部分 咕噜译 20141128 00:17:16
- Line部分 咕噜译 2014-11-29 04:34:09
- Symbol 部分 咕噜译 2014-11-29 8:00
- Chord 部分 咕噜译 2014-11-29 8:00
- Area部分 Harry译 20140419 咕噜校对 2014-11-29 10:18:17
- Diagonal部分 魏飞 译 20140418 17.14.18咕噜校对 2014-11-29 11:06:40
- Arc部分 魏飞 译 20140716 19-25咕噜校对2014-11-29 10:49:56

> Wiki ▸ [[API--中文手册]] ▸ [[SVG函数]] ▸ **SVG 轴**

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

D3的轴组件自动展示比例尺参照的线。这使得你可以专注于展示数据，而轴组件需要关心绘制坐标轴和刻度标记的繁琐任务。

Axis 轴组件设计来和D3的定量，时间和序数比例尺一起使用。

d3.svg.axis()

创建一个默认的轴。

axis(selection)

将轴应用到选择和过渡上。选择必须包含svg或者g元素。例如：
d3.select("body").append("svg") .attr("class", "axis") .attr("width", 1440) .attr("height", 30) .append("g") .attr("transform", "translate(0,30)") .call(axis);

axis.scale([scale])

如果指定了scale 参数则设置刻度尺，并返回轴。如果未指定scale参数，将返回当前的刻度尺，默认为线性刻度。

axis.orient([orientation])

如果指定了方向orientation 参数则设置orientation ，并返回轴。如果未指定orientation参数，将返回当前的刻度尺，默认为“bottom”。支持下面几种方向：• top -刻度位于横轴域路径上面 • bottom -刻度位于横轴域路径下面 • left -刻度位于纵轴域路径左边 • right -刻度位于纵轴域路径右边 如果指定的方向是不支持的值之一，该轴将恢复为默认的方向。改变方向将影响刻度和它们的标签相对于轴路径的的位置，但不改变该轴本身的位置; 为了改变轴相对于基址图的位置，可以指定g元素上的transform 变换属性。

axis.ticks([arguments…])

如果指定了arguments 参数，存储指定的参数为之后来用生成刻度并返回轴。参数之后会传递给scale.ticks生成刻度值（除非刻度值通过明确地指定axis.tickValues）。参数将传递给比例尺的tickFormat方法用来生成默认的刻度格式。如果没有指定参数，返回当前的刻度参数，默认是[10]。

合适的参数取决于关联的比例尺：对于线性比例尺，你可以指定刻度数为axis.ticks(20)；对于对数比例尺你可以指定数量和刻度格式；对于时间比例尺，时间间隔例如axis.ticks(d3.time.minutes, 15)可能更合适。

axis.tickValues([values])

如果指定了values 数组，指定的values 将用于刻度，而不是使用使用比例尺的自动刻度生成器。如果values 是null，清空任何预先设定的明确的刻度值，回到原来比例尺的生成器。如果没有指定values ，返回当前设定的刻度值，默认是null。例如，为了生成刻度为指定的values ： var xAxis = d3.svg.axis() .scale(x) .tickValues([1, 2, 3, 5, 8, 13, 21]);

明确地刻度值优先于通过使用axis.ticks设置刻度参数。但是，任何的参数都仍然传递给比例尺的tickFormat 函数，如果一个刻度格式也没设置；这样，设置axis.ticks和axis.tickValues它可能有效。

axis.tickSize([inner, outer])

如果指定了inner和outer，设置内部和外部刻度尺寸为指定的值并返回轴。如果inner和outer没有指定，返回当前的内部刻度尺寸，默认是6。

axis.innerTickSize([size])

如果指定了size ，设置内部刻度尺寸为指定的值并返回轴。如果没有指定size ，返回当前的内部刻度尺寸，默认是6。内部刻度尺寸控制刻度线的长度，从轴的原生位置偏移。

axis.outerTickSize([size])

如果指定了size ，设置外部刻度尺寸为指定的值，并返回轴。如果没有指定size ，返回当前的外部刻度尺寸，默认是6。外部刻度尺寸控制域路径末尾的平方长度，从轴的原生位置偏移。这样，因此，“外刻度”实际上不是刻度但是域路径的一部分，并且它们的位置由相关的比例尺的域范围来确定。这样，外刻度可能与第一个或最后内部刻度重叠。外刻度尺寸0的禁止域路径的平方端，而不是产生一条直线。

axis.tickPadding([padding])

如果指定填充边距，设置填充边距的指定值并返回对应的axis。如果没有指定填充边距，返回当前默认填充边距（默认为3像素）。

axis.tickFormat([format])

如果指定格式，格式设置为指定的函数并返回axis。如果没有指定格式，返回当前格式函数，默认为空。空格式表示应该使用比例尺的默认格式器，此格式通过调用scale.tickFormat产生。在这种情况下，指定的参数同样传递给scale.tickFormat方法。 查看d3.format创建格式的帮助。例如,axis.tickFormat(d3.format(.0f“))将通过逗号分组千位显示一个整数。首先定义格式器：var commasFormatter = d3.format(”,.0f”) 可以让你把它作为你的数据的函数，例如,在comma-grouped整数前添加“$”符号：.tickFormat(function(d) { return "$" + commasFormatter(d); })。 注意:对于对数比例尺，刻度的数值不能自定义；然而，刻度的数值标签可以通过ticks自定义。同样，对数比例尺刻度的格式器通常是通过ticks而不是tickFormat指定，以保持默认label-hiding行为。

- axis.scale axis.orient小屁孩翻译，咕噜校正20140425 21:40:39
- axis.tickPadding axis.tickFormat魏飞20140427译 咕噜校对 2014-11-29 17:03:41
- 其余 咕噜译 2014-11-29 16:56:20

> Wiki ▸ [[API--中文手册]] ▸ [[SVG函数]] ▸ **SVG 控件**

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

d3.svg.brush()

构造一个新的刷子，使用默认的x和y比例尺，和空的范围extent。

brush(selection)

绘制或重绘当前brush拖选到指定的选择元素；brush可以同时绘制多个元素，值得注意的是，这些选择刷会共用相同的背景范围；通常一个选择刷一个时间只能绘制一个元素；selection 参数可以是一个变换，在这种情况下，选择刷将自动执行变换；可以使用brush.event 来触发选择刷事件在动画刷的变换过程中。

brush.x([scale])

获取或设置选择刷相关联的x比例尺；如果指定scale， 则设置x比例尺为指定的scale 并返回brush，如果未指定scale， 则返回当前的x比例尺，默认为null；变换通常可指定为定量比例尺，在这种情况下，范围extent处于比例尺域 domain的数据空间；然而，它也可以被定义为定量比例尺代替，这时，范围extent来自于比例尺的的range extent的像素空间。

brush.y([scale])

获取或设置选择刷相关联的y比例尺；如果指定scale， 则设置y比例尺为指定的scale 并返回brush，如果未指定scale， 则返回当前的y比例尺，默认为null；比例尺通常可指定为定量比例尺，在这种情况下，extent是来自于比例尺域 domain的数据空间；然而，它也可以被定义为序数比例尺ordinal scale代替，这时，范围extent来自于变换的range extent的像素区间。

brush.extent([values])

获取或设置当前选择刷的范围，如果指定values，则设置范围为指定的值并返回当前brush；如果未指定values，则返回当前的范围；范围的定义依赖于关联的比例尺；如果x和y比例尺都可用，范围是一个二维的数组：[[x0, y0], [x1, y1]]，x0和y0是范围的最低端，x1和y1是范围的

最顶端；如果只有x比例尺可用，范围被定义为一维数组：[x0, x1]，同样地，如果只有y变换可用，范围被定义为：[y0, y1]；如果没有变换可用，范围为null。 当范围被设定为指定的值values，所得到的范围会被正确的保存起来；然而，一旦选择刷被用户移动（鼠标按下并被拖动），这时，范围必须要调用 scale.invert来重新计算；注意，在这种情况下，值可能由于像素的精度有限而略有偏差。 注意，这并不会自动重绘选择刷或触发任何的监听事件；想要重绘选择刷，可以在选择器或过渡上调用 brush，想要触发事件，使用 brush.event。

brush.clamp([clamp])

设置或获取当前的夹选行为，如果指定clamp，则设置夹选行为为指定值并返回brush，如果未指定clamp，则返回当前的行为；夹选行为的定义依赖于关联的比例尺；如果x和y比例尺都可用，夹选行为是一个数组[x,y]，x和y是布尔类型，用来确定是否二维范围内每个维度应该被夹选到各自的x和y比例尺；如果只有x或y比例尺可用，夹选行为是一个布尔类型，用来指定是否一维范围该被夹选到比例尺，如果变换都不可用，则夹选行为是null。

brush.clear()

Clears the extent, making the brush extent empty. 清空范围，使得brush的范围为 empty。

brush.empty()

当且仅当选择刷的范围为空时，返回true；当brush被创建时，被初始化为空；当点击背景而不移动时，或者范围为空时，选择刷会变为空的；如果选择刷有零宽度或零高度，它将被视为空；当选择刷为空，则它的范围即视为未定义。

brush.on(type[, listener])

设置或获取指定类型type的监听器listener ；选择刷支持三种类型事件： • Brushstart - 鼠标按下时，即mousedown； • brush - 鼠标移动时，如果范围在改变，即mousemove； • brushend – 鼠标弹起/松开时，即mouseup； 需要注意，当鼠标在背景上点击时也会触发brush事件，因为选择刷范围会立刻被清除来开始一段新的范围。

brush.event(selection)

如果selection是选择器，立刻触发brush行为到注册的监听器，即三个事件序列： brushstart, brush 和 brushend；这是非常有用的，在设置完 brush extent后来触发相应的事件；如果selection 是一个过渡，注册合适的补间动画，这样在过渡的过程中来触发事件：当过渡开始于初始设置范围时触发brushstart， 过渡进行期间每刻都会触发brush， 过渡结束时触发brushend ；需要注意，当用户开始刷时，即使过渡没结束也会被立刻终止interrupted。

- 魏飞译 2014-07-25 19:25 咕噜校对 2014-11-29 20:06:46

> Wiki ▸ [[API Reference]] ▸ **Time**

See one of:

- [[Time Formatting]] - convert between times and strings.
- [[Time Scales]] - compute visual encodings of times.
- [[Time Intervals]] - perform simple time arithmetic.

> Wiki ▸ [[API Reference]] ▸ [[Time]] ▸ **Time Formatting**

D3 includes a helper module for parsing and formatting dates modeled after the venerable strptime and strftime C-library standards. These functions are also notably available in Python's time module.

d3.time.**format**(*specifier*)

Constructs a new local time formatter using the given *specifier*. (Equivalent to locale.timeFormat for the default U.S. English locale.) The specifier string may contain the following directives.

- `%a` - abbreviated weekday name.
- `%A` - full weekday name.
- `%b` - abbreviated month name.
- `%B` - full month name.
- `%c` - date and time, as "%a %b %e %H:%M:%S %Y".
- `%d` - zero-padded day of the month as a decimal number [01,31].
- `%e` - space-padded day of the month as a decimal number [1,31]; equivalent to `%_d`.
- `%H` - hour (24-hour clock) as a decimal number [00,23].
- `%I` - hour (12-hour clock) as a decimal number [01,12].
- `%j` - day of the year as a decimal number [001,366].
- `%m` - month as a decimal number [01,12].
- `%M` - minute as a decimal number [00,59].
- `%L` - milliseconds as a decimal number [000, 999].
- `%p` - either AM or PM.
- `%S` - second as a decimal number [00,61].
- `%U` - week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
- `%w` - weekday as a decimal number [0(Sunday),6].
- `%W` - week number of the year (Monday as the first day of the week) as a decimal number [00,53].
- `%x` - date, as "%m/%d/%Y".
- `%X` - time, as "%H:%M:%S".
- `%y` - year without century as a decimal number [00,99].
- `%Y` - year with century as a decimal number.
- `%Z` - time zone offset, such as "-0700".
- `%%` - a literal "%" character.

For %U, all days in a new year preceding the first Sunday are considered to be in week 0. For %W, all days in a new year preceding the first Monday are considered to be in week 0. In some implementations of strftime and strptime (as in Python), a directive may include an

optional field width or precision; this feature is not yet implemented in D3, but may be added in the future.

For locale-specific date and time formatters, see locale.timeFormat.

The % sign indicating a directive may be immediately followed by a padding modifier:

- `0` - zero-padding
- `_` - space-padding
- `-` - disable padding

If no padding modifier is specified, the default is `0` for all directives, except for `%e` which defaults to `_`).

The returned *format* is both an object and a function. For example:

```
var format = d3.time.format("%Y-%m-%d");
format.parse("2011-01-01"); // returns a Date
format(new Date(2011, 0, 1)); // returns a string
```

format(*date*)

Formats the specified *date*, returning the corresponding string. The *date* must be a JavaScript Date object.

```
var monthNameFormat = d3.time.format("%B");
var dayNameFormat = d3.time.format("%A");
monthNameFormat(new Date(2014, 4, 1));  //returns string "May" (remember javascript month
dayNameFormat(new Date(2014, 4, 1));  //returns string "Thursday"
```

Note that when dates are used in conjunction with quantitative scales, the dates are implicitly coerced to numbers representing the number of milliseconds since UNIX epoch. To convert between numbers and dates, you can use the following code:

```
time = +date; // convert a Date object to time in milliseconds
date = new Date(time); // convert a time in milliseconds to a Date object
```

If you prefer to be explicit, you can also use the date object's getTime method, but the + operator is shorter and possibly faster.

format.**parse**(*string*)

Parses the specified *string*, returning the corresponding date object. If the parsing fails, returns null. Unlike "natural language" date parsers (including JavaScript's built-in parse), this method is strict: if the specified string does not exactly match the associated format specifier, this method returns null. For example, if the associated format is the full ISO 8601

string "%Y-%m-%dT%H:%M:%SZ", then the string "2011-07-01T19:15:28Z" will be parsed correctly, but "2011-07-01T19:15:28", "2011-07-01 19:15:28" and "2011-07-01" will return null, despite being valid 8601 dates. (Note that the hard-coded "Z" here is different from `%Z`, the time zone offset.) If desired, you can use multiple formats to try multiple format specifiers sequentially.

The `%d` and `%e` format specifiers are considered equivalent for parsing.

d3.time.format.**multi**(*formats*)

Returns a new multi-resolution time format given the specified array of predicated *formats*. Each format is a two-element array consisting of a format specifier string (such as that passed to the d3.time.format constructor) and a predicate function. For any date that is passed to the returned time format, the first predicate function that returns true will determine how the specified date is formatted. For example, the default time format used by d3.time.scale is implemented as:

```
var format = d3.time.format.multi([
  [".%L", function(d) { return d.getMilliseconds(); }],
  [":%S", function(d) { return d.getSeconds(); }],
  ["%I:%M", function(d) { return d.getMinutes(); }],
  ["%I %p", function(d) { return d.getHours(); }],
  ["%a %d", function(d) { return d.getDay() && d.getDate() != 1; }],
  ["%b %d", function(d) { return d.getDate() != 1; }],
  ["%B", function(d) { return d.getMonth(); }],
  ["%Y", function() { return true; }]
]);
```

Thus, if the specified date is not a round second, the milliseconds format (`".%L"`) is used; otherwise, if the specified date is not a round minute, the seconds format (`":%S"`) is used, and so on. See bl.ocks.org/4149176 for an example.

The **multi** method is available on any d3.time.format constructor. For example, d3.time.format.utc.multi returns a multi-resolution UTC time format, and locale.timeFormat.multi returns a multi-resolution time format for the specified locale.

d3.time.format.**utc**(*specifier*)

Constructs a new UTC time formatter using the given *specifier*. (Equivalent to locale.timeFormat.utc for the default U.S. English locale.) The specifier may contain the same directives as the local time format. Internally, this time formatter is implemented using the UTC methods on the Date object, such as getUTCDate and setUTCDate in place of getDate and setDate.

d3.time.format.**iso**

The full ISO 8601 UTC time format: "%Y-%m-%dT%H:%M:%S.%LZ". Where available, this method will use Date.toISOString to format and the Date constructor to parse strings. If you depend on strict validation of the input format according to ISO 8601, you should construct a time format explicitly instead:

```
var iso = d3.time.format.utc("%Y-%m-%dT%H:%M:%S.%LZ");
```

> Wiki ▸ [[API--中文手册]] ▸ [[比例尺]] ▸ 时间比例尺

- 本文档是D3官方文档中文翻译，并保持与最新版同步。
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

D3的Time scale是d3.scale.linear比例尺的扩展，使用Javascript的Date对象作为其输入域（domain）。因此，不同于普通的线性比例尺（linear scale），domain的值会被强制转为时间类型而非数字类型；同样的，invert函数返回的是一个时间类型。最方便的是，time scale同样提供了基于time intervals的合适的打点器，解除了为任何基于时间的域生成轴的痛苦。d3.time.scale返回的是一个比例尺对象，同时也是一个函数。你可以像任何函数一样引用它，同时它有额外的方法可以改变它的行为。如D3的其它类一样，scale对象也可以链式调用方法，setter方法返回scale对象本身，可以用一个简洁的声明中引用多个setters。

d3.time.scale()

使用默认的域和范围来构建一个新的时间比例尺；默认以本地时间配置ticks和tick format。

d3.time.scale.utc()

使用默认的域和范围来构建一个新的时间比例尺；默认以UTC时间配置ticks和tick format。

scale(x)

给定输入域内的时间x，返回相应输出范围的值；

scale.invert(y)

对输出范围y内的值，返回相应的输入域x内的时间对象。这个用于从rang到domain的反向映射。对于输出域range内的y值，scale(scale.invert(y))==y；同样，对于输入域domain内的时间对象x，scale.invert(scale(x))==x。这种反向操作符在交互时非常有用，例如，鼠标移动时，计算出鼠标当前位置相对应的输入域的时间。

scale.domain([dates])

如果设置了参数dates，则把scale的输入域domain设置为dates数组，该数组必须包含两个及以上的时间对象。如果数组里的元素不是时间对象，将被强制转换为时间对象；这个强制转换在比例尺被引用时发生。如果没有设置参数dates，则返回scale的当前的输入域domain。通常来说，时间比例尺虽然在输入域domain里只有两个时间对象，但你可以为polylinear 比例尺设置多于两个的时间对象，这样的话，必须在输出范围中设置相同数量的数值。

scale.nice([interval[, step]])

scale.nice([count])

扩展域使得开始和结束于很好的整数值，当由指定的时间间隔和可选参数阶数step。作为替代方案，指定一个明确的时间间隔，一个数字count 可以被指定，并且在时间间隔将自动被选择为与scale.ticks一致。如果未指定计数count ，则默认为10。 这个方法通常扩展比例尺的域，可能只扩展边界到最接近的整数值。Nicing是有用的，如果域是从数据计算的并可以是不规则的。例如，对于域[2009-07-13T00:02, 2009-07-13T23:48]好域是[2009-07-13, 2009-07-14]。如果域有多于两个值，nicing将只影响域的第一个和最后一个值。

scale.range([values])

如果设置了参数values，则把scale的输出域range设置为values数组，该数组必须包含两及以上的数值，以匹配输入域domain。values数组里的元素不需要一定是数字类型，任何支持下面的interpolator的数据类型都可以。但是，invert操作符需要数字类型的输出域。如果没有设置参数values，则返回scale的当前输出域。

scale.rangeRound([values])

设置scale的输出域range为指定的values数组，同时设置了scale的interpolator为d3.interpolateRound。如果比例尺输出的值应该是整数时，这是一个便捷的方法，可以有效地避免平滑处理工件。也可以在引用比例尺应用之后手动四舍五入输出的值。

scale.interpolate([factory])

如果设置了参数factory，则将scale的输出插值器设置为factory。默认的插值器工厂为d3.interpolate，将标准化的[0,1]范围内的输入域参数t，映射到相应的输出域range内的数值。插值器工厂将为输出范围内的相邻的每对数值构造插值器。

scale.clamp([boolean])

如果设置了参数boolean（布尔值），则相应地开启或者关闭clamping。默认情况下，clamping是关闭的，如果一个超出了输入域的值被传递给了比例尺，则比例尺将通过线性推断，可能会返回一个输出范围之外的数值。以默认的[0,1]输入域和输出域为例，输入数值2将返回输出数值2。如果clamping开启，标准化的输入域参数t会被限定在输出域[0,1]内，这样比例尺返回的值永远都在输出域内。如果boolean没有设置，返回的是比例尺是否限定当前数值到输出范围内。

scale.ticks([interval[, step]])

scale.ticks([count])

返回比例尺输入域的代表性时间。返回的打点tick时间，它们有相同间距（模式不标准的时间间隔，比如月份和闰年）、包含可读性的数值（比如午夜），以及确保在输入域范围之内。刻度常被用在显示参考线、刻度标记，与视觉化的数据一起使用时。 如果参数count是一个数字，将返回大约count数量的刻度。如果未设置参数count，默认设置为10。Count只是一个提示，根据输入域domain，比例尺可能会返回更多或更少的数值。 如果设置了参数时间间隔interval ，那么将会引用time interval的range function 函数，同时传递可选参数step，以生成刻度。 比如，生成默认的10个刻度 scale.ticks(10); 或者，以15分钟为间隔生成ticks : scale.ticks(d3.time.minute, 15); 注意：对于UTC 比例尺，使用相应的UTC输出范围方法（比如，d3.time.minute.utc）。: 以下时间间隔被视为自动ticks： 1-, 5-, 15- 和 30-秒. 1-, 5-, 15- 和 30-分. 1-, 3-, 6- 和 12-小时. 1- 和 2-天. 1-星期. 1- 和 3-月. 1-年. 这一组时间间隔是稍微有点随意的，并且额外的值也会在未来会被加上。

scale.tickFormat(count)

返回一个可以显示刻度值的世时间格式化函数。参数count应该和用于生成刻度值的数量一致。你不用必须使用比例尺内建的刻度格式，但它会根据输入域的时间自动计算相应的结果。 比如以下时间格式： %Y – 年分界，如"2011". %B – 月分界，如"February". %b %d – 星期分界，如"Feb 06". %a %d – 日期分界，如"Mon 07". %I %p – 小时分界，如"01 AM". %I:%M – 分钟分界，如"01:23". :%S – 秒分界，如":45". .%L – 毫秒分界，如".012". 使用多重时间格式，默认的时间格式为每个时间间隔提供局部和全局环境。例如，为显示[11 PM, Mon 07, 01 AM]，刻度格式器可以同时展现小时和日期信息--而非只有小时。如果你喜欢用单一的时间格式器，你可以一直使用你自己定义的 d3.time.format。

scale.copy()

返回当前time scale的完整副本。所有对当前世家比例尺的改变都不会改变返回的副本，反之亦然。 何凯琳译 20141124 gulu校 2014-11-29

> Wiki ▸ [[API Reference]] ▸ [[Time]] ▸ **Time Intervals**

Time intervals are irregular! For example, there are 60 seconds in a minute, but 24 hours in a day. Even more confusing, some days have 23 or 25 hours due to daylight saving time, and the standard Gregorian calendar uses months of differing lengths. And then there are leap years!

To simplify manipulation of and iteration over time intervals, D3 provides a handful of time utilities in addition to the time scale and format. The utilities support both local time and UTC time. Local time is determined by the browser's JavaScript runtime; arbitrary time zone support would be nice, but requires access to the Olson zoneinfo files.

Interval

d3.time.*interval*

Returns the specified *interval*. The following intervals are supported:

- d3.time.second
- d3.time.minute
- d3.time.hour
- d3.time.day
- d3.time.week (alias for d3.time.sunday)
- d3.time.sunday
- d3.time.monday
- d3.time.tuesday
- d3.time.wednesday
- d3.time.thursday
- d3.time.friday
- d3.time.saturday
- d3.time.month
- d3.time.year

interval(*date*)

Alias for *interval*.floor(*date*). For example, `d3.time.day(new Date())` returns midnight (12:00 AM) on the current day, in local time.

interval.**floor**(*date*)

Rounds down the specified *date*, returning the latest time interval before or equal to *date*. For example, `d3.time.day.floor(new Date())` returns midnight (12:00 AM) on the current day, in local time.

interval.**round**(*date*)

Rounds up or down the specified *date*, returning the closest time interval to *date*. For example, `d3.time.day.round(new Date())` returns midnight (12:00 AM) on the current day if it is on or before noon, and midnight of the following day if it is after noon.

interval.**ceil**(*date*)

Rounds up the specified *date*, returning the earliest time interval after or equal to *date*. For example, `d3.time.day.ceil(new Date())` returns midnight (12:00 AM) on the following day, in local time (unless you happen to run this code at exactly midnight, in which case it returns the current time).

interval.**range**(*start*, *stop*[, *step*])

Returns every time interval after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th interval will be returned, based on the interval number (such as day of month for d3.time.day). For example, a *step* of 2 will return the 1st, 3rd, 5th *etc.* of the month with d3.time.day.

interval.**offset**(*date*, *step*)

Returns a new date equal to *date* plus *step* intervals. If *step* is negative, then the returned date will be before the specified *date*; if *step* is zero, then a copy of the specified *date* is returned. This method does not round the specified *date* to the interval. For example, if it is currently 5:34 PM, then `d3.time.day.offset(new Date(), 1)` returns 5:34 PM tomorrow (even if Daylight Savings Time changes!).

interval.**utc**

Returns a corresponding time interval in UTC rather than local time. For example, `d3.time.day.range(start, stop)` returns local time days between *start* and *stop*, while `d3.time.day.utc.range(start, stop)` returns UTC days between *start* and *stop*.

Intervals

d3.time.**second**

Seconds (e.g., 01:23:45.0000 AM). Always 1,000 milliseconds long.

d3.time.**minute**

Minutes (e.g., 01:02:00 AM). ECMAScript explicitly ignores leap seconds, so minutes are always 60 seconds (6e4 milliseconds) long.

d3.time.**hour**

Hours (e.g., 01:00 AM). 60 minutes long (36e5 milliseconds). Note that advancing time by one hour can return the same hour number, or skip an hour number, due to Daylight Savings Time.

d3.time.**day**

Days (e.g., February 7, 2012 at 12:00 AM). Most days are 24 hours long (864e5 milliseconds); however, with Daylight Savings Time, a day may be 23 or 25 hours long.

d3.time.**week**

Alias for d3.time.sunday. A week is always 7 days, but ranges between 167 and 169 hours depending on Daylight Savings Time.

d3.time.**sunday**

Sunday-based weeks (e.g., February 5, 2012 at 12:00 AM).

d3.time.**monday**

Monday-based weeks (e.g., February 6, 2012 at 12:00 AM).

d3.time.**tuesday**

Tuesday-based weeks (e.g., February 7, 2012 at 12:00 AM).

d3.time.**wednesday**

Wednesday-based weeks (e.g., February 8, 2012 at 12:00 AM).

d3.time.**thursday**

Thursday-based weeks (e.g., February 9, 2012 at 12:00 AM).

d3.time.**friday**

Friday-based weeks (e.g., February 10, 2012 at 12:00 AM).

d3.time.**saturday**

Saturday-based weeks (e.g., February 11, 2012 at 12:00 AM).

d3.time.**month**

Months (e.g., February 1, 2012 at 12:00 AM). Ranges between 28 and 31 days.

d3.time.**year**

Years (e.g., January 1, 2012 at 12:00 AM). Normal years are 365 days long; leap years are 366.

Aliases

d3.time.**seconds**(*start*, *stop*[, *step*])

Alias for d3.time.second.range. Returns the second boundaries (e.g., 01:23:45 AM) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th second will be returned, based on the second of the minute. For example, a *step* of 15 will return 9:01:45 PM, 9:02:00 PM, 9:02:15 PM, *etc.*

d3.time.**minutes**(*start*, *stop*[, *step*])

Alias for d3.time.minute.range. Returns the minute boundaries (e.g., 01:23 AM) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th minute will be returned, based on the minute of the hour. For example, a *step* of 15 will return 9:45 PM, 10:00 PM, 10:15 PM, *etc.*

d3.time.**hours**(*start*, *stop*[, *step*])

Alias for d3.time.hour.range. Returns the hour boundaries (e.g., 01 AM) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th hour will be returned, based on the hour of the day. For example, a *step* of 3 will return 9 PM, 12 AM, 3 AM, *etc.*

d3.time.**days**(*start*, *stop*[, *step*])

Alias for d3.time.day.range. Returns the day boundaries (midnight) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th date will be returned, based on the day of the month. For example, a *step* of 2 will return the 1st, 3rd, 5th *etc.* of the month.

d3.time.**weeks**(*start*, *stop*[, *step*])
d3.time.**sundays**(*start*, *stop*[, *step*])
d3.time.**mondays**(*start*, *stop*[, *step*])
d3.time.**tuesdays**(*start*, *stop*[, *step*])
d3.time.**wednesdays**(*start*, *stop*[, *step*])
d3.time.**thursdays**(*start*, *stop*[, *step*])
d3.time.**fridays**(*start*, *stop*[, *step*])
d3.time.**saturdays**(*start*, *stop*[, *step*])

Aliases for d3.time.*interval*.range etc. Returns the week boundaries (midnight Sunday) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th week will be returned, based on the week of the year. For example, a *step* of 4 will return January 2, January 30, February 27, *etc.*

d3.time.**months**(*start*, *stop*[, *step*])

Alias for d3.time.month.range. Returns the month boundaries (e.g., January 01) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th month will be returned, based on the month of the year. For example, a *step* of 3 will return January, April, July, *etc.*

d3.time.**years**(*start*, *stop*[, *step*])

Alias for d3.time.year.range. Returns the year boundaries (midnight January 1st) after or equal to *start* and before *stop*. If *step* is specified, then every *step*'th year will be returned. For example, a *step* of 5 will return 2010, 2015, 2020, *etc.*

Counting

d3.time.**dayOfYear**(*date*)

Returns the day number for the given date. The first day of the year (January 1) is always the 0th day. Unlike the d3.time.format's %j directive, dayOfYear is 0-based rather than 1-based.

d3.time.**weekOfYear**(*date*)
d3.time.**sundayOfYear**(*date*)
d3.time.**mondayOfYear**(*date*)
d3.time.**tuesdayOfYear**(*date*)
d3.time.**wednesdayOfYear**(*date*)
d3.time.**thursdayOfYear**(*date*)
d3.time.**fridayOfYear**(*date*)
d3.time.**saturdayOfYear**(*date*)

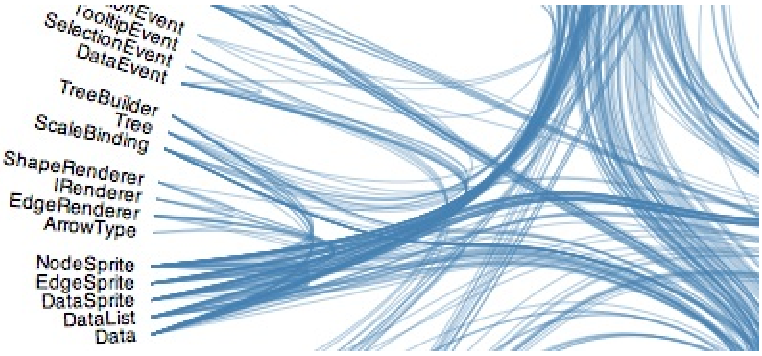
Returns the week number for the given date, where weeks start with the given *day*. The first day of the year (January 1) is always the 0th week. weekOfYear is an alias for sundayOfYear, which is equivalent to d3.time.format's %U directive. mondayOfYear is equivalent to d3.time.format's %W directive.

> Wiki ▸ [[API--中文手册]] ▸ 布局

- [[Bundle|捆绑布局]] - 对边使用Holten 层次捆绑算法。
- [[Chord|弦布局]] - 从关系矩阵生成一个弦图。
- [[Cluster|簇布局]] - 将实体聚集成树状图。
- [[Force|力布局]] - 模拟物理力排放相连节点的位置。
- [[Hierarchy|层次布局]] - 这是个抽象布局，可派生一个定制的层次布局。
- [[Histogram|直方图布局]] - 使用量化的箱计算数据的分布。
- [[Pack|包布局]] - 用递归的圆形包装生成一个层次布局。
- [[Partition|分区布局]] - 递归地将节点树分割为旭日状或者冰柱状。
- [[Pie|饼布局]] - 计算饼图或圆环图中弧的开始和结束角度。
- [[Stack|堆叠布局]] - 计算堆叠图或者面积图的基线。
- [[Tree|树布局]] - 整齐地排列树节点。注意簇布局不是整齐的。
- [[Treemap|矩形树布局]] - 使用空间递归分区算法展示树的节点。

> Wiki ▸ [[API Reference]] ▸ [[Layouts]] ▸ **Bundle Layout**

Implements Danny Holten's hierarchical edge bundling algorithm. For each input link, a path is computed that travels through the tree, up the parent hierarchy to the least common ancestor, and then back down to the destination node. This sequence of nodes can then be used in conjunction with other hierarchical layouts, such as cluster to generate bundled splines between nodes:



For example, consider this visualization of software dependencies.

d3.layout.**bundle**()

Constructs a new default bundle layout. Currently, the bundle layout is stateless and thus only has a default configuration. The returned layout object is both an object and a function. That is: you can call the layout like any other function, and the layout has additional methods that change its behavior. Like other classes in D3, layouts follow the method chaining pattern where setter methods return the layout itself, allowing multiple setters to be invoked in a concise statement.

bundle(*links*)

Evaluates the bundle layout on the specified array of *links*, returning the computed path from the source to the target, through the least common ancestor. Each input link must have two attributes:

- source - the source node.
- target - the target node.

Furthermore, each node must have one attribute:

- parent - the parent node.

This is a subset of the fields generated by the hierarchy layouts. The return value of the layout is an array of paths, where each path is represented as an array of nodes. Thus, the bundle layout does not compute the basis splines directly; instead, it returns an array of

nodes which implicitly represent the control points of the spline. You can use this array in conjunction with d3.svg.line or d3.svg.line.radial to generate the splines themselves. For example, if you were to use a cluster:

```
var cluster = d3.layout.cluster()
    .size([2 * Math.PI, 500]);
```

A suitable line generator for hierarchical edge bundling might be:

```
var line = d3.svg.line.radial()
    .interpolate("bundle")
    .tension(.85)
    .radius(function(d) { return d.y; })
    .angle(function(d) { return d.x; });
```

The bundle layout is designed to work in conjunction with the line generator's "bundle" interpolation mode, though technically speaking you can use any interpolator or shape generator. Holten's bundle strength parameter is exposed as the line's tension.

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 弦布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

弦图所示内容为一组实体之间的关系。例如，假定有一组不同发色的人：黑色、金色、棕色和红色。该组每个人都为约会对象准备了中意的发色；在所有29630个（假设）黑发人群中，40%的人（11975）倾向于同具有相同发色的对象进行约会。但这种倾向是非对称的：例如，只有10%的金发人会选择黑发人群作为约会对象，而20%的黑发人会将金发人作为约会对象。

通过在不同的弧线之间画出二次贝塞尔曲线，将上述关系表示在一张弦图中。源弧线和目标弧线分别代表总人口的两个镜像子集，如喜欢金发的黑发人口数量，以及喜欢黑发的金发人口数量。另一个例子，我们看这个软件依赖弦图的例子：http://bl.ocks.org/mbostock/1046712。 弦布局同弦形和弧形协同工作，用于生成数据对象。该对象作为弦形状的输入，对弦进行描述。同时，该布局还能生成对不同群组的描述，用作弧形的输入。

d3.layout.chord()

构建新的弦布局。在默认情况下，输入数据并未分类，并且各群组之间没有填充。和其他布局不同，该弦布局并不是应用于数据的函数；相反，数据通过设置关联矩阵来指定，通过chords和groups访问器检索。

chord.matrix([matrix])

指定矩阵之后，设定该布局用到的输入数据矩阵。如果没有指定矩阵，返回当前数据矩阵，默认为未定义。输入矩阵的数字必须为“方形矩阵”(http://en.wikipedia.org/wiki/Matrix_(mathematics\)#Square_matrices)例如：[[11975,5871,8916,2868], [1951,10048,2060,6171], [8010,16145,8090,8045], [1013,990,940,6907]]

矩阵的每一行对应一个特定分组，如上文所述某个发色。矩阵中每一列i同第i行相对应；每个单元格ij对应表示第i组到第j组之间的关系。

chord.padding([padding])

If padding is specified, sets the angular padding between groups to the specified value inradians. If padding is not specified, returns the current padding, which defaults to zero. You may wish to compute the padding as a function of the number of groups (the number of rows or columns in the associated matrix). 指定填充之后，在不同组之间设定角度填充，为指定的值（弧度为单位）。如果没有指定填充，返回当前填充，默认值为0。你可能希望计算填充是分组数量（关联矩阵中行和列的数量）的函数。

chord.sortGroups([comparator])

如果已经指定comparator，使用指定comparator函数为布局设定分组（行）的排列顺序。为每两行调用comparator函数，传递的入参是行i和行j的总和。通常，需要将comparator按照d3.ascending或d3.descending进行指定。如果没有指定comparator，则返回当前分组排列顺序，该顺序默认值为空。

chord.sortSubgroups([comparator])

如果已经指定comparator，使用指定comparator函数为布局设定分组（行内各列）的排列顺序。为每对单元格调用comparator函数，值为各单元格的值。通常，需要将comparator以升序或降序进行指定。如果没有指定comparator，则回当前子分组排列顺序，该顺序默认值为空。

chord.sortChords([comparator])

如果已经指定comparator，运用指定comparator函数为弦布局设定弦（Z顺序）的排列顺序。为每两条弦调用comparator函数，入参为源单元格和目标单元格的最小值。通常，要将comparator以升序或降序进行指定。如果没有指定comparator，返回当前chord排列顺序，默认值为空。

chord.chords()

给定布局的当前配置和关联矩阵，返回计算过的弦对象。如果弦对象已计算完毕，本方法返回缓存值。如果布局属性有任何改变，则清空之前计算的弦。此时，如果下次调用该方法，需要对布局进行重新计算。返回对象具有下列属性： source -描述源对象。 target -描述目标对象。 这两个对象描述下列实体： index -行索引，i。 subindex索引-列索引，j。 startAngle-弧的起始角，在radians内。 endAngle-弧的终止角，在radians内。 value -关联单元格ij的数值。 需要注意的是，这些对象同弦很方便为弦生成器匹配默认的访问器；但仍可以对访问器进行重写或者修改返回对象，实现布局微调。

chord.groups()

给定布局的当前配置和关联矩阵，返回计算过的分组对象。如果分组对象已计算完毕，本方法返回缓存值。如果布局属性有任何改变，则清空之前计算的分组。此时，如果下次调用该方法，需要对布局进行重新计算。返回对象具有下列属性： index -行索引，i。 startAngle -弧的起始角，在radians内。 endAngle -弧的终止角，在radians内。 value -相关行 i的值的总和。 需要注意的是，这些对象同弧度生成器的默认访问器具有较好的吻合度；但仍可以对访问器进行重写或者修改返回对象，实现布局微调。

- 张烁译 20140428
- 咕噜校对 2014-11-30 09:39:57

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 簇布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

簇布局可以产生树状图：将树的叶节点放在同一深度的节点连接图。例如，簇布局可以用来在软件包层次结构中组织类：

同D3中的其他类一样，布局遵循方法链模式，在该模式下setter方法返回布局本身，允许使用简单语句调用多个setter。

d3.layout.cluster()

使用默认设置创建新的簇布局：默认排序为空；默认子访问器假定每个输入数据为带子数组的对象；默认分离函数在同层级使用一个节点宽度，在不同层级使用两个节点宽度；默认尺寸为1×1。

cluster(root)

cluster.nodes(root)

运行簇布局，返回节点数组及指定的根节点。簇数组为D3分层布局家族的一部分。这些布局具有相同的基本结构：布局的输入参数为分层的根节点，输出返回值为一个数组，表示计算过的所有节点的位置。每个节点都有各自属性：

• 上层-父节点，在根节点时为空。 • 下层-子节点数组，在叶节点时为空。 • 深度-节点深度，从根节点计算，值从0开始。 • x-节点位置的x坐标值。 • y-节点位置的y坐标。

虽然布局在x和y轴有值存在，但这表示一个任意坐标系；例如，可以将x坐标视为直径，将y坐标当做角，从而形成一个射线，而非笛卡尔坐标系布局。

cluster.links(nodes)

指定节点数组，如以节点形式返回的数组，返回对象节点表示每个节点中父节点同子节点之间的关系。叶节点没有任何关系。每个节点都是一个具有两个属性的对象： source -父节点（如上述所示）。 target -子节点。 该方法在获取一组关系描述时很有效果，通常与对角图形生成器共同使用。例如： svg.selectAll("path") .data(cluster.links(nodes)) .enter().append("path") .attr("d", d3.svg.diagonal());

cluster.children([children])

如果子节点已经指定，则设定子节点访问器函数。如未指定，则返回当前子节点访问器函数，该函数将输入数据默认为带子数组的对象： functionchildren(d){ returnd.children; }

通常，使用d3.json可以方便地加载节点分层，并将输入分层表示为一个嵌套JSON对象。例如： { "name":"flare", "children":[{ "name":"analytics", "children":[{ "name":"cluster", "children":[{"name":"AgglomerativeCluster","size":3938}, {"name":"CommunityStructure","size":3812}, {"name":"MergeEdge","size":743}] }, { "name":"graph", "children":[{"name":"BetweennessCentrality","size":3534}, {"name":"LinkDistance","size":5731}] }] }] }

在分层中，子访问器在根节点首先被调用。如果访问器返回值为空，则该节点在布局遍历结束时被假定为叶节点。否则，访问器需要返回数据元素数组，来表示子节点。

cluster.sort([comparator])

如已指定comparator，则使用指定的comparator函数设置布局中同级节点排序顺序。如comparator未指定，则返回当前分组的排序顺序，默认值为空。为每对节点，调用comparator函数。 comparator的默认值为空，此时采用三种遍历顺序，排序被禁用。例如，按照输入数据的字符串名对同层节点以降序顺序排序，即： functioncomparator(a,b){ returnd3.ascending(a.name,b.name); } See d3.ascending or d3.descending for details. 详见：d3.ascending或d3.descending

cluster.separation([separation])

如果已经指定separation，使用指定的函数计算相邻节点的间距。如果未指定separation，则返回当前间距函数。默认情况下，该函数为： 存在一个更加适合于射线布局的变动，可以根据直径大小相应减少间距： functionseparation(a,b){ return(a.parent==b.parent?1:2)/a.depth; }

两个相邻节点a和b传递到separation函数，且必须返回节点间期望的间距。节点通常是同级的，虽然，这些节点也可能属于相近关系（或更远的关系），如果布局将这些节点临近放置的话。

cluster.size([size])

如果已经指定size，则将可用布局尺寸设定为指定的二元数组，以x和y来表示。如果尺寸没有指定，则返回当前尺寸，默认值为1×1，如果nodeSize正在使用中，则默认值为空。虽然布局在x和y轴都有数值，但该坐标系可以是任意坐标系。例如，创建一个射线布局，其中树的广度（x）用角度来测量, 树的深度（y）表示半径r, 单位为像素，即 [360, r]。

cluster.nodeSize([nodeSize])

如果nodeSize已经指定，以二元数组x和y的形式返回每个节点的固定尺寸。如果nodeSize没有指定，则返回当前节点尺寸，默认值为空，表示布局尺寸总体固定，可以使用size来获得。

cluster.value([value])

如果已经指定value，则用指定的函数设定值访问器。如果尚未指定value，则返回当前值访问器，默认值为空，表示值属性没有计算。指定之后，每次输入数据元，都会调用值访问器，并且必须返回一个用以表示节点数值的数字。该值对集群布局没有影响，但它是分层布局所提供的通用功能。

- 张烁译 20140430
- 咕噜校对 2014-11-30 10:42:08

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 力布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

一个灵活的力导向图布局实现使用位置Verlet整合算法允许简单地控制。有关物理模拟的更多信息，请参见Thomas Jakobsen。这个实例是利用四叉树加速电荷相互作用使用Barnes–Hut approximation。除了电荷charge力，pseudo-gravity力保持节点中心在可见区域并且避免排出断开子图，当链接的固定距离的几何约束时。额外的自定义力和约束可应用于"tick" 事件,简单地通过更新节点的x和y属性。 http://bl.ocks.org/mbostock/4062045

一些例子： • divergent forces：http://bl.ocks.org/mbostock/1021841 • multiple foci：http://bl.ocks.org/mbostock/1021953 • graph constructor：http://bl.ocks.org/mbostock/929623 • force-directed tree：http://bl.ocks.org/mbostock/1062288 • force-directed symbols：http://bl.ocks.org/mbostock/1062383 • force-directed images and labels：http://bl.ocks.org/mbostock/950642 • force-directed states：http://bl.ocks.org/mbostock/1073373 • sticky force layout：http://bl.ocks.org/mbostock/3750558

像D3的其他类，布局遵循方法链模式setter方法返回布局自身。允许使用一个简单的声明调用多个setter方法。有别于其他的布局实现是无状态的。力导向图布局 内部保持关联节点和链接的引用。因此，一个给定的力导向图布局实例只可以和一个单一的数据集一起使用。

d3.layout.force()

构造一个新的力导向布局使用默认设置：尺寸 1×1，链接长度1，摩擦0.9，距离20，充电强度-30，重力强度0.1和θ参数0.8。默认的节点和链接是空数组，并且当布局开始时，内部的α冷却参数被设置成0.1。构建力导向图布局的通用模式是所有配置属性, 然后调用开始start函数：

var force = d3.layout.force() .nodes(nodes) .links(links) .size([w, h]) .linkStrength(0.1) .friction(0.9) .distance(20) .charge(-30) .gravity(0.1) .theta(0.8) .alpha(0.1) .start();

需要注意的是, 像D3的其他布局，力导向图布局不要求特定的可视化表现。最常见的，节点被映射到SVG圆形元素，链接被映射为SVG线元素。但你也可以显示节点作为符号或图片。. 显示为符号的案例：http://bl.ocks.org/mbostock/1062383 显示为图片的案例：http://bl.ocks.org/mbostock/950642

force.size([size])

如果指定了size，设置可用的布局大小为指定的代表x和y的两元素数字数组来。如果未指定size，返回当前size，默认为1×1。size影响力导向图的两个方面：重力中心和初始的随机位置。重心是 [x / 2, y / 2]。当节点被添加到力导向图布局，如果不具有已设置的x和y属性，然后这些属性都分别使用范围为[0, x]和[0, y]的均匀随机分布进行初始化。

force.linkDistance([distance])

如果指定了distance，设定链接节点间的目标距离为指定的值。如果未指定distance，返回布局的当前链路距离，默认为20。如果distance 是常量，那么所有的链接是相同的距离。否则，如果距离是一个函数，则该函数为每个链接（按顺序）求值，被传递参数是链接和它的索引，用this上下文作为力导向图布局；该函数的返回值被用来设置每个链接距离。当布局开始时该函数被求值。通常, 距离被指定以像素为单位; 然而，这些单位也可以是相对于布局的size.

链接不实现为在一般力导向图布局中的"弹性力"，而是弱几何约束。对于布局的每一个tick，计算每对链接节点之间的距离并与目标距离进行比较；链接随后朝向彼此移动或远离彼此，以收敛到所需的距离。这种在Verlet 集成算法之上约束松弛的方法大大稳定于之前使用弹性力的方法，并且还允许在Tick事件监听器灵活实现其他约束，如层次分层。

force.linkStrength([strength])

如果指定了strength，设置链接间强度（刚性）为[0,1]范围内的指定的值。如果strength 未被指定，返回布局的当前链接强度，默认为1。如果强度是一个常数，那么各个链接都有相同的强度。否则，如果强度是一个函数，那么该函数为每一个链接（按顺序）求值，传递的参数是链接和它的索引，this 上下文是这个力布局；该函数的返回值随后被用来设置每个链接的强度。每当布局开始时调用这个函数。

force.friction([friction])

如果指定了friction，设定摩擦系数为指定的值。如果摩擦系数未被指定，返回当前系数，默认为0.9。这个参数的名称可能有误导性的；它不对应标准物理的摩擦系数。相反，它更接近速度衰减：在模拟的每个tick，粒子速度通过指定的friction缩减。因此，值1对应于无摩擦的环境中，而一个0值冻结所有颗粒就位。超出范围的值[0,1]不推荐，可能有破坏稳定的的影响。

force.charge([charge])

如果指定电荷强度charge，设置电荷强度charge 为指定的值。如果电荷强度charge 未被指定，返回电流电荷强度，其默认值为-30。如果电荷强是常量，那么所有节点都具有相同的电荷。否则，如果电荷强度是一个函数，则该函数为每个节点（按顺序）求值，传递的参数是节点和它的索引，this上下文作为力布局；该函数的返回值被用于设置每个节点的电荷强度。每当布局开始时被调用。.

负值导致节点排斥，而正值导致节点吸引。对于图形布局，应使用负值；对于N体模拟，可以使用正值。所有节点都假定为无穷远的小点具有相等电荷和质量。电荷力是通过Barnes–Hut算法高效实现的，为每个tick计算四叉树。电荷力设置为零禁用四叉树，它可以显着提高性能，如果你不需要N体模拟。 Barnes–Hut算法：http://arborjs.org/docs/barnes-hut

force.chargeDistance([distance])

如果distance 被指定，设置电荷强度已经应用的最大距离。如果distance 未被指定，返回当前最大电荷距离，默认为无穷大。指定一个有限电荷距离提高力导向图的性能和产生更本地化的布局。限定距离的电荷力是尤其有用当合自定义重力一起使用时。有关示例, 请参阅“Constellations of Directors and their Stars” (The New York Times). 限定距离的电荷力：http://www.nytimes.com/newsgraphics/2013/09/07/director-star-chart/

force.theta([theta])

为了避免大图的二次性能下降。力导向图布局使用Barnes–Hut逼近，每个tick花费O(n log n)。对于每一个tick，创建一个四叉树用于存储当前节点的位置；然后，对每个节点，计算给定节点的所有其他节点的总电荷力。为了聚集过远的节点，通过处理节点的距离的簇作为单个逼近电荷力。Theta 确定计算精度：对一个节点到块象限中心的距离，四叉树中象限区域的比率小于theta，在给定象限的所有节点被视为一个单一的，大的节点，而不是单独地计算。Barnes–Hut近似：http://en.wikipedia.org/wiki/Barnes-Hut_simulation

force.gravity([gravity])

如果指定了重力gravity，设置引力强度为指定的值。如果未指定重力，返回当前的引力强度，默认为0.1。这个参数的名称可能是误导性的：它不对应于物理重力gravity（可以用一个正电荷参数进行仿真)。相反，重力被实现为类似于虚拟弹力的每个节点连接到布局尺寸的中心的弱几何约束。这种方法具有很好的特性：靠近布局的中心，引力强度几乎为零，避免了布局的任何局部变形；当节点将被推远离中心，引力强度与距离成线性比例变强。因此，重力总是会某个阈值克服斥力电荷势力，以防止断开连接的节点逃逸出布局。 重力可以通过设置引力强度为0来禁止。如果禁用重力，建议你实现一些其他的几何约束，以防止逃逸布局，如在布局的范围内制约它们的节点。

force.nodes([nodes])

如果节点nodes 被指定，设置布局的相关节点为指定的nodes 数组。如果未指定节点nodes ，则返回当前数组，默认为空数组。每个节点具有以下属性：

• index - nodes 数组节点的索引（从零开始）。 • x -当前节点的x坐标位置。 • y -当前节点的位置y坐标。 • px -前一个节点位置的x坐标。 • py -前一个节点位置的y坐标。 • fixed -一个布尔值，表示节点位置是否被锁定。 • weight -节点权重；相关联的链接的数目。 这些属性不必在传递节点给布局之前进行设置；如果他们都没有设置，合适的默认值将在布局进行初始化start时调用，但是，要知道，如果你的节点上存储有其他数据，你的数据属性不应该与上面使用的布局属性冲突。

force.links([links])

如果指定了链接links ，设置布局的相关链接为指定的links 数组。如果没有指定链接links 时，返回当前数组，默认为空数组。每个链接都有以下属性： • source - 源节点（节点中的元素）。. • target - 目标节点（节点中的元素）。 注意：在源和目标属性的值可初始化为nodes 数组的索引；这些将被替换为调用开始函数之后的引用。链接对象可能有你指定的其他字段，这个数据可以用来计算链接强度strength和距离distance，基于每个连接的基础使用一个访问函数

force.start()

启动模拟；当首次创建布局时此方法必须被调用，然后分配节点和链接。此外，每当节点或链接发生变化它应当再次调用。在内部，布局使用冷却参数alpha控制布局的温度：当物理模拟收敛为稳定的布局，温度就下降，造成节点移动速度比较慢。最终，alpha下降到低于阈值，模拟完全停止，释放CPU资源，避免电池电量的消耗。布局可以使用恢复或重新启动重新加热；使用拖曳的行为时，会自动出现这种情况。

在开始时，布局初始化相关节点上的各种属性。每个节点的索引是通过遍历数组，从零开始计算。初始的x和y坐标，如果尚未在外部设置为以有效的数字，通过检测相邻节点计算：如果链接的节点已经在x或y的初始位置时，相应的坐标被施加到新节点。这当新节点被添加时增加图形布局的稳定性，而不是使用默认值（在布局的尺寸之内随机初始化位置）。前一px和py位置设置为初始位置，如果尚未设置，给新节点一个零初始速度。最后，固定布尔默认为false。

布局还在相关链接上初始化源source 和目标target 属性：为方便起见，这些属性可以被指定为一个数字索引，而不是直接的链接，使得节点和链接可以从JSON文件或其他静态的描述中读取。如果这些属性是数字，导入链接的源和目标属性仅替换为nodes 中相应的实体；因

此，现有链接中的这些属性当布局被重新启动时不受影响。链接距离distances和强度strengths也在开始时重新计算 .

force.alpha([value])

获取或设置力布局的冷却参数：alpha。如果值value 已指定，设置alpha为指定的值并返回力布局。如果值大于零，这个方法也将重新启动力布局（如果它尚未运行），分发一个“启动”事件启用节拍定时器。如果值为非正，且力布局正在运行，这个方法将在下一个tick停止力布局并分派“结束”事件。如果未指定值，则该方法返回当前alpha值。

force.resume()

相当于： force.alpha(.1); 设置冷却参数alpha为0.1。此方法设置内部的alpha参数设置为0.1，然后重新启动定时器。 通常情况下，你不需要直接调用此方法；它是通过start自动调用。它也可以通过拖动动作drag自动调用。

force.stop()

Equivalent to: force.alpha(0); Terminates the simulation, setting the cooling parameter alpha to zero. This can be used to stop the simulation explicitly, for example, if you want to show animation or allow other interaction. If you do not stop the layout explicitly, it will still stop automatically after the layout's cooling parameter decays below some threshold. 相当于: force.alpha(0); 终止模拟，冷却参数alpha设定为零。这可以用来显式地停止模拟，例如，如果你要展示的动画或允许其他的互动。如果你没有明确停止布局，它仍然会自动在布局的冷却参数衰变后低于某个阈值后停止。

force.tick()

执行力布局仿真一步。这种方法可以在配合使用start and stop来计算静态布局。例如：force.start(); for (var i = 0; i < n; ++i) force.tick(); force.stop(); 迭代次数取决于图形的大小和复杂性。初始位置的选择也可以对如何快速将图形收敛于一个很好的解产生显着影响。例如，下面的节点沿对角线排列： var n = nodes.length; nodes.forEach(function(d, i) { d.x = d.y = width / n * i; }); 如果不手动初始化位置，力布局将它们随机初始化，导致有些不可预知的行为。

force.on(type, listener)

注册指定的监听器listener 为力布局指定类型的事件。目前，仅支持"start"，"tick"和"end"事件。"tick"事件将被指派为模拟的每个tick。监听节拍事件来更新节点和链接的显示位置。例如，如果你最初显示的节点和链接，象这样：

var link = vis.selectAll("line") .data(links) .enter().append("line");

var node = vis.selectAll("circle") .data(nodes) .enter().append("circle") .attr("r", 5); You can set their positions on tick: force.on("tick", function() { link.attr("x1", function(d) { return d.source.x; }) .attr("y1", function(d) { return d.source.y; }) .attr("x2", function(d) { return d.target.x; }) .attr("y2", function(d) { return d.target.y; });

node.attr("cx", function(d) { return d.x; }) .attr("cy", function(d) { return d.y; }); }); 您可以在tick设置他们的位置: 在这个案例中，我们已经存储选择的node和link在初始化中，这样我们就不需要重新选择每个节点的tick。如果你愿意，你可以不同地显示节点和链接；例如，您可以使用符号而不是圆形。 当模拟内部的alpha冷却参数达到零，"end"事件就被调度。

force.drag()

绑定一个行为允许交互式拖动到节点，无论是使用鼠标或触摸。和节点中的call操作符一起使用；例如，初始化时调用node.call(force.drag)。拖动事件在鼠标滑过时设置节点的固定属性，这样，只要鼠标移动到某个节点，它停止不动。鼠标滑过而不是鼠标按下时固定，使得它更容易捕捉移动节点。当接收到一个鼠标按下事件在每个后续的鼠标移动直到鼠标弹起，当前鼠标位置设置为节点的中心。此外，每个鼠标移动触发一个力布局重新开始，再加热模拟。如果你想拖拖之后节点保持固定，在头动开始时设置fixed 属性为true，如粘力布局的例子。

实现注意：在鼠标移动和鼠标弹起事件监听器已注册当前窗口上，这样，当用户开始拖动节点，他们可以继续拖动节点，即使鼠标离开窗口。每个事件监听器使用“force”命名空间，以避免和其他你可能想绑定到节点或窗体上的事件监听器冲突。如果节点被拖动事件移除了，随后的点击事件将被抓获到的最后一个鼠标抬起事件触发，你可以忽略这些点击和拖动通过查看是否默认的行为被阻止了。 selection.on("click", function(d) { if (d3.event.defaultPrevented) return; // ignore drag otherwiseDoAwesomeThing(); }); See the collapsible force layout and divergent forces for examples.

咕噜译 2014-11-30 20:37:37

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 层次布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

分层布局是一种抽象布局，不直接使用。但它允许在不同的分层布局中共享节点。请看下列例子： • Cluster -簇实体构成树状图。 • Pack -使用递归圆填充法创建一个分层布局。 • Partition -将节点树递归划分成辐射状或环状。 • Tree -将节点树整齐放置。 • Treemap -使用递归空间划分显示节点树。 虽然不是分层布局，但捆绑布局可以同分层共同使用。

d3.layout.hierarchy()

使用以下默认设置创建新的分层布局：默认排序顺序为值的降序排列；默认值访问器假定每个输入数据为一个含数值属性的对象；默认子访问器假定每个输入数据为一个含子数列的对象。

hierarchy(root)

运行分层布局，返回节点数组及指定的根节点。布局的输入参数为分层的根节点，输出返回值为一个数组，表示计算过的所有节点的位置。每个节点都有各自属性： parent -父节点，在根节点时为空。 children -子节点数组，在叶节点时为空。 value -节点值，由值访问器返回。 depth -节点深度，从根节点计算，值从0开始。 此外，多数分层布局也会计算节点的x和y的位置；详见实现类。

hierarchy.links(nodes)

指定节点数组返回一个对象数组，该数组表示每个节点中父节点同子节点之间的关系。叶节点没有任何关系。每个节点都是一个具有两个属性的对象： 源-父节点（如上述所示）。 目标-子节点。 该方法在获取一组关系描述时很有效果，通常与对角图形生成器共同使用。例如： svg.selectAll("path") .data(partition.links(nodes)) .enter().append("path") .attr("d",d3.svg.diagonal());

hierarchy.children([accessor])

如果accessor已经指定，则设定子访问器函数。如未指定，则返回当前子访问器函数，该函数将输入数据默认为带子数组的对象： functionchildren(d){ returnd.children; }

通常，使用d3.json可以方便地加载节点分层，并将输入分层表示为一个嵌套JSON对象。例如： { "name":"flare", "children":[{ "name":"analytics", "children":[{ "name":"cluster", "children":[{"name":"AgglomerativeCluster","size":3938}, {"name":"CommunityStructure","size":3812}, {"name":"MergeEdge","size":743}] }, { "name":"graph", "children":[{"name":"BetweennessCentrality","size":3534}, {"name":"LinkDistance","size":5731}] }] }] } 在分层中，子访问器在根节点首先被调用。如果访问器返回值为空，则该节点在布局遍历结束时被假定为叶节点。否则，访问器需要返回数据源数组，来表示子节点。参数node和depth都需要调用访问器。

hierarchy.sort([comparator])

如已指定comparator，则使用指定的comparator函数设定布局的同级节点节点顺序。如comparator未指定，则返回当前分组的排序顺序，默认值为按照输入数据的字符串名对节点的降序顺序排序：

functioncomparator(a,b){ returnb.value-a.value; }

为每对节点，调用comparator函数。零comparator禁用排序，使用树遍历顺序。comparator函数也可以通过d3.ascending或d3.descending实现。

hierarchy.value([value])

如果已经指定value，则用指定的函数设定值访问器。如果尚未指定value，则返回当前值访问器。默认访问器假定输入数据为一个具有数值属性的对象： functionvalue(d){ returnd.value; }

每次输入数据元素，都会调用值访问器，并且必须返回一个用以表示节点数值的数字。对于区域布局，如树图，该值用于设定每个节点值相应的面积。对于其他布局，该值对簇布局没有影响。

hierarchy.revalue(root)

对于一棵指定的树，从根开始重计算每个节点的值，但不需对子节点进行重新排序或重新计算。该方法可以用于对每个节点值进行重新计算，但又不必对分层做出任何结构改变。最初，该方法是用来支持sticky treemaps的。

- 张烁译 20140430
- 咕噜校对 2014-11-30 10:42:08

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 直方图布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

直方图布局可以用来表示数据分布，通过将离散数据点分组归纳到箱子里。使用实例详见bl.ock 3048450。

d3.layout.histogram()

使用默认值访问器、范围函数和箱函数，构建新的直方图函数。默认条件下，直方图函数返回值为频率。返回布局对象既是一个对象，也是一个函数。即: 可以像调用其他函数一样调用该布局，并且布局有额外的方法改变自身行为。和D3中的其他类一样，布局遵循方法链模式，在该模式下setter方法返回布局本身，允许使用简单语句调用多个setter。

histogram(values[, index])

在指定的values数组上计算直方图。可以指定一个可选参数index, 传递给范围函数和箱函数。返回值为数组的数组：外部数组的每个元素表示一个容器，每个容器包含输入values的相关元素。此外，每个容器有三个属性：

• x -箱的下界（包含）。 • dx -箱的宽度；x + dx为上界（不包含）。 • y - the count (if frequency is true), or the probability (if frequency is false).计数（如果frequency为true），或概率（如果frequency为假）。 请注意，在频率方式上，y属性和长度属性相同。

histogram.value([accessor])

指定从关联数据中提取值的方法；accessor是一个函数，每当输入值传递到histogram时，都需要调用该函数，即等于在计算直方图之前调用values.map(accessor)。默认值函数为内置Number，与恒等函数类似。如果未指定accessor，则返回当前值访问器。

histogram.range([range])

指定直方图范围。忽略在指定范围之外的值。可以通过二元数组指定range，数组表示范围的最大值和最小值；或者将range指定为一个函数，该函数返回values数组和传递到histogram的当前索引。默认范围为值的长度（minimum和maximum）。如果未指定range，则返回当前范围函数。

histogram.bins()

histogram.bins(count)

histogram.bins(thresholds)

histogram.bins(function)

详细说明如何将值归类到直方图中。如果没有指定参数，则返回当前箱函数，默认值为Sturges' formula的一个实现，Sturges' formula使用等间隔的值将值划分到不同的箱当中。如果已经指定count值，则将range的值均匀分布到指定数量的箱中。 如果已指定thresholds数组，则它定义了箱的极限值，从最左边的值（最小值）开始到最右边的值（最大值）。n + 1 thresholds指定了n个箱。任何小于thresholds[1]的值都将被放在第一个箱中；同理，任何大于或等于thresholds[thresholds.length - 2]的值将被放在最后一个箱中。因此，虽然第一个和最后一个极值并未分配到箱中，但他们对于定义第一个箱的x属性和最后一个箱的dx属性还是有必要存在的。 最后，如果已经指定箱function，该函数会在布局传递数据时调用，传递当前range，值得数列和当前索引传递到histogram。该函数必须返回上文所述的thresholds数列。

histogram.frequency([frequency])

指定直方图的y值是否是一个计数（频率）或概率（密度）；默认值为频率。如果没有指定频数，则返回当前频率的布尔值。 张烁译 20140430 咕噜校对 2014-11-30 10:42:08

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 包布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

下图采用包含（嵌套）来展现层级结构。每一个叶子节点的大小都显示了每个数据点数量的大小。如图所示，通过小圈的大小累积逐渐接近于一个大圈，但要注意的是，由于空间上的浪费，在层级之间也会存在一些失真，但只要叶子节点可以精确的比较就行。尽管圆形填充没有像treemap那样，高效率的使用了空间，空间上的浪费反而清晰地展现了分层效果。

通过展开层级，pack layout 也可以被用来创建bubble charts（气泡图表）。

和D3中其他类相似，layout中的布局遵循方法链模式，其中setter方法返回布局本身，并允许在一个简单语句中调用多个setter方法。

d3.layout.pack()

使用默认设置创建一个新的包布局，默认的排序顺序是按值升序排序；默认的子访问器，假设每一个输入数据都是一个带有子数组的对象，默认的大小是1×1。

pack(root)

pack.nodes(root)

运行包布局，返回与指定根节点root相关联的节点的数组。簇布局是D3家族分层布局中的一部分。这些布局遵循相同的基本结构：将输入参数传递给布局是层次的根节点root， 输出的返回值是一个代表所有节点计算出的位置的数组。每个节点上填充以下几个属性： • parent – 父节点，或根节点为null。 • children – 子节点数组，或叶子节点为null。 • value – 节点的值，作为访问器返回的值。 • depth – 节点的深度，根节点从0开始。 • x – 计算的节点位置的x坐标。 • y – 计算的节点位置的y坐标。 • r – 计算的节点半径。

pack.links(nodes)

给定一个特定节点的数组nodes，例如由节点返回的，返回表示每个节点的从父母到孩子链接的对象数组。而叶子节点将不会有任何的链接。每个链接都是一个对象，且具有两个属性： • source – 父节点（正如上述描述的那样）。 • target – 子节点。 此方法检索一组适合展示的链接描述很有用，通常与对角线形状发生器一起用。例如： svg.selectAll("path") .data(cluster.links(nodes)) .enter().append("path") .attr("d", d3.svg.diagonal());

pack.children([children])

如果指定children 参数，则设置特定的孩子访问器函数。如果children没有指定，则返回当前孩子访问函数，默认假定输入的数据是一个带有孩子数组的对象： function children(d) { return d.children; } 通常情况下，可以使用d3.json很方便加载节点层次结构，代表输入层次结构的嵌套JSON对象。例如：

{ "name": "flare", "children": [{ "name": "analytics", "children": [{ "name": "cluster", "children": [{"name": "AgglomerativeCluster", "size": 3938}, {"name": "CommunityStructure", "size": 3812}, {"name": "MergeEdge", "size": 743}] }, { "name": "graph", "children": [{"name": "BetweennessCentrality", "size": 3534}, {"name": "LinkDistance", "size": 5731}] }] }] } 孩子的访问器是第一次被层次中的根节点调用。如果存取器返回null，则该节点被假定为叶节点并且布局遍历终止。否则，访问器应返回表示子节点的数据元素的数组。

pack.sort([comparator])

如果comparator 比较器指定了，则使用特定的比较器函数，设置同级节点布局的排序顺序。如果comparator 没有指定，则返回当前组排序顺序，默认为升序排列，按通过相关的输入数据的数值属性来排序：

function comparator(a, b) { return a.value - b.value; }

这个比较器的函数被节点对调用，输入的数据传递给每个节点。空比较器禁止排序，使用树遍历的顺序。比较器函数也可以使用d3.ascending或d3.descending来实现排序。

pack.value([value])

如果value 值指定了，则设置值访问器为指定的函数。如果value 值未指定，则返回当前的值访问器，其中假定输入数据是一个带有一个数字值属性的对象：

function value(d) { return d.value; } 对每一个输入的数据元素调用值访问器，并且必须返回一个数字，来表示该节点的数值。此值被用来按比例设置每个圆的面积为value。但是请注意，圆圈大小是严格地只在叶节点之间才存在可比性；内部的节点无法准确地比较，因为在打包子圆圈和他们父母之间还留有空白。

pack.size([size])

如果指定了大小size，则设定可用的布局大小为指定的表示x和y的二元数组。如果size 大小没有指定，则返回当前大小，默认为1×1。

pack.radius([radius])

如果指定了半径radius，则设置半径的函数，用于计算每个节点的半径。如果半径radius为空，因为默认情况下，半径自动从节点值确定，且调整为适合的布局大小。如果半径radius未指定，则返回当前的半径函数，默认为null。此半径radius 也可为均匀的圆大小指定为一个恒定数目。

pack.padding([padding])

如果指定padding，则设置相邻圈之间的大概填充，以像素为单位。如果没有指定padding，则返回当前的填充，默认为0。

- 何凯琳译 2014-12-01
- gulu校对2014-12-6 21:32:15

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 层次布局 ▸ 分区布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

分区布局将会产生邻接的图形：一个节点链的树图的空间填充转化体。节点将被绘制为实心区域图（无论是弧还是矩形），而不是在层次结构中绘制父子间链接，节点将被绘制成固定区域（弧度或者方形），并且相对于其它节点的位置显示它们的层次结构中的位置。节点的尺寸将编码为一个量化的维度，这将难以在一个节点链图中展示。

就像D3的其他类，布局将遵循方法链模式，setter方法返回布局本身，并通过一个简洁的语句来调用多个setter方法。

d3.layout.partition()

创建一个新的带有默认设置的分区布局：默认为值降序的排序顺序；默认值的访问器假定为每一个输入值是一个带有数值属性的对象；默认子值的访问器假定为每一个输入值是一个带有子值数组的对象，大小为1为1×1。

partition(root)

partition.nodes(root)

运行分区布局，将返回指定根节点的相关联的节点阵列数组。分区布局是D3家族分层布局中的一部分。这些布局将遵循相同的基本结构：传递给布局的输入参数值是层次结构的根节点，并且输出返回值将是一个代表所有节点经过计算的位置的数组。每个节点上将拥有以下几个属性：

• parent - 父节点，或空的根节点。 • children - 子节点的阵列数组，或者为空的叶节点。 • value - 该节点的值，值访问器所返回的值。 • depth - 节点的深度（即节点的层级数），根节点为0。 • X–节点位置的最小x坐标。 • y–节点位置的最小y坐标。 • dx - 节点位置的x范围。 • dy - 节点位置的y范围。

虽然布局只有一个x和y尺寸，但它却可以表示一个任意的坐标系；例如，你可以把X作为半径、y作为角度用以产生径向而非笛卡尔布局。在笛卡尔取向上，x，y，dx和dy分别相当于SVG矩形元素的“x”，“y”，“width”和“height”属性。在径向取向上，它们可以被用于计算弧生成器的innerRadius，startAngle，outerRadius和endAngle。笛卡尔取向可被称为冰柱树，而径向取向被称为旭日图。

partition.links(nodes)

给定指定的节点数组，比如这些返回的节点，将返回一个对象数组，该数组表示从父到子的每个节点的链接。叶节点将不会有任何的链接。每个链接是一个对象，每个对象具有以下两个属性： • source - 父节点（如上所述）。 • target - 子节点。 此方法适用于检索一组适于显示的链接描述，通常与对角线形状生成器结合使用。例如： svg.selectAll("path") .data(partition.links(nodes)) .enter().append("path") .attr("d", d3.svg.diagonal());

partition.children([children])

如果指定了children，将设置指定的children访问器方法。如果没有指定children，则返回当前的children访问器方法，默认输入数据是一个带有一个children数组的对象： function children(d) { returnd.children; }

通常情况下，使用d3.json可以很方便地装入节点层次结构，并且将输入层次结构作为嵌套JSON对象。例如： { "name": "flare", "children": [{ "name": "analytics", "children": [{ "name": "cluster", "children": [{"name": "AgglomerativeCluster", "size": 3938}, {"name": "CommunityStructure", "size": 3812}, {"name": "MergeEdge", "size": 743}] }, { "name": "graph", "children": [{"name": "BetweennessCentrality", "size": 3534}, {"name": "LinkDistance", "size": 5731}] }] }] }

children访问器将在层次结构根节点中首先被调用。如果访问器返回null，则该节点被认为为叶节点，直到布局遍历终止。否则，访问器就会返回一个表示子节点的数据元素的数组。

partition.sort([comparator])

如果指定的comparator，将使用指定的比较方法来设置为布局的同级节点的排序顺序。如果没有指定comparator，返回当前组的排序顺序，其默认为相关的输入数据的数值属性降序排列： function comparator(a, b) { returnb.value - a.value; }

比较器方法用来被节点对调用，此节点对传递给每个节点的输入数据。若比较器方法为null，则排序将被禁用，之后将使用树的遍历顺序。比较器方法也可以使用d3.ascending或d3.descending实现。

partition.value([value])

如果指定value，便将其设置为指定的访问器方法。否则返回当前值访问器，即输入数据是一个带有数值属性的对象： function value(d) { returnd.value; }

值访问器将被每个输入的数据元素调用，并且一定会返回一个数字，该数字表示该节点的数值属性值。此值将被成比例的设置每个节点的面积值。

partition.size([size])

如果指定的size，将通过指定的二元素数组[x,y]设置为有效布局的大小。如果没有指定size，返回当前布局大小，默认为1×1。

- Harry 译 2014-11-29
- 咕噜校对 2014-11-30 21:02:50

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 饼布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

饼布局方便与计算组成饼图或圈图的弧的开始和结束的角度：

饼布局并不是只能绘制饼状图，如果你喜欢，你还可以直接用她来绘制弧形状。饼布局会简单的将一个数据数组（如字符串数据）转换成一个对象数据，这个对象数组中会包含开始角度和结束角度，这些角度的范围为0到2π，然后传给弧形生成器。

d3.layout.pie()

构造一个新的饼图函数，使用默认的值访问器（数值）、进行比较排序（降序）、开始角度（0）、结束角度（2π）；返回的布局对象即是对象也是一个函数，这就意味着：你可以想调用方法一样调用该布局对象，布局还具有改变其行为的其他方法；就像D3的其他类型方法一样，饼布局也支持方法链模式，setter方法返回布局自身，允许在一个简单的申明中调用多个setter方法。

pie(values[, index])

用指定的values数组计算饼函数。一个可选的index 参数会被传递给开始和结束的函数；返回值是一个数组，数组元素包含以下属性：• value - 数据值，计算来自于value 生成器；• startAngle - 弧的开始弧度，非角度；• endAngle - 弧的结束弧度，非角度；• data - 数据原生的值； 按照原生的排序返回元素，匹配values参数，即使排序顺序已经应用了；这保证了数组中每个元素原生的索引，这是非常好的，当你使用原生的值数组来匹配颜色分类时。

pie.value([accessor])

指定如何从关联的数据提取值（如：给饼图绑定一个访问器函数）；accessor 是一个函数，会在每个输入值传给pie时调用，原理相当于在计算饼布局前，先调用了values.map(accessor)；该函数可以传两个参数：当前的数据d和所在索引i；默认的值函数是内置的Number，类似于特征函数；如果未指定accessor，则返回当前的访问器。

pie.sort([comparator])

如果指定comparator，则设置数据的排序方法为指定的；传入null来禁止排序；如果未指定comparator，则返回当前的comparator ；默认的排序方法是降序；排序会保留原生数据的索引，只会影响到角度；排序函数会在每对传给 pie的数据上调用；当然，该排序函数你可以使用 d3.ascending 或 d3.descending代替。

pie.startAngle([angle])

如果指定angle，则设置饼布局所有的起始弧度为指定值：如果未指定angle，返回当前的值，默认为0；起始弧度可以被指定为常数或一个函数，如果是函数，当 pie被调用时，会进行一次起始弧度的计算，被传递的参数有：当前数据d和索引i。

pie.endAngle([angle])

如果指定angle，则设置饼布局所有的起始弧度为指定值：如果未指定angle，返回当前的值，默认为2π；起始弧度可以被指定为常数或一个函数，如果是函数，当 pie被调用时，会进行一次起始弧度的计算，被传递的参数有：当前数据d和索引i。

- 魏飞译 2014-07-16-19-25
- 咕噜校对 2014-11-30 21:19:34

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 堆叠布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

堆叠布局需要一个二维的数据数组，并计算基准线；这个基准线会被传到上层，以便生成一个堆叠图。支持多个基线算法，以及启发式的排序算来可以提高感知灵敏度，就像拜伦(Byron)和瓦腾伯格(Wattenberg)在“Stacked Graphs—Geometry & Aesthetics”（http://www.leebyron.com/else/streamgraph/download.php?file=stackedgraphs_byron_wattenberg.pdf）中所说的那样。

堆叠布局可以工作在任意的二维x，y坐标系空间，就像是D3的其他布局一样，包括树布局(tree)。因此，图层可以被垂直、水平叠放，或者是其他径向的叠放(radially)。尽管图表的默认偏移是零，但是依然可以使用扭动或摆动(wiggle)的偏移量来绘制流图，它会尽量的减少在偏移时所产生的锯齿边界。

d3.layout.stack()

构造一个新的堆叠布局，使用默认的偏移（零）和排序（空null）；返回的布局对象是一个对象也是一个函数；也就是说：你可以向调用函数一样调用布局，布局是具备改变其行为的方法的；和D3中其他类相似，布局遵循方法链模式，其中setter方法返回布局本身，并允许在一个简单语句中调用多个setter方法。

stack(layers[, index])

为各层计算y坐标的基线，并传到相应的层中。最简单的情况，层是一个二维的值数组；所有的第二维的数组必须是相同的长度。y和x访问器被用来分别定义每层的x坐标位置的y方向厚度；因此下面这些值都是必须的：• x - x位置处所对应的值，即x坐标；• y - y处厚度所对应的值；• y0 - y方向最小的y值，即基线； 以上这些属性可以通过重写访问器或out函数进行自定义。

stack.values([accessor])

指定在每一层中如何从相关联的元素中提取值，访问器是一个函数，并被传递给层以被调用在每一个输入层上，相当于是在计算堆叠布局前调用了layers.map(accessor)。默认的值函数是内置对象，类似于标识函数。如果未指定访问器accessor，则返回当前的值访问器。 值访问器可以被用于关联每层额外的数据，而不仅仅是每一个点；例如，假设你的数据结构如下： var layers = [{ "name": "apples", "values": [{ "x": 0, "y": 91}, { "x": 1, "y": 290}] }, {

"name": "oranges", "values": [{ "x": 0, "y": 9}, { "x": 1, "y": 49}] }]; 指定一个值访问器来检索每层的点： var stack = d3.layout.stack() .offset("wiggle") .values(function(d) { return d.values; }); 然后，如果你想给每层添加一个tooltip，你可以这样写： svg.selectAll("path") .data(stack(layers)) .enter().append("path") .attr("d", function(d) { return area(d.values); }) .append("title") .text(function(d) { return d.name; });

stack.offset([offset])

如果指定offset，则设置堆叠的偏移算法为指定的offset ；如果未指定offset，则返回当前的offset；以下的字符串值可以被使用： • silhouette - 居中流，类似于 ThemeRiver； • wiggle - 尽量减少斜率变化比例； • expand -标准化层以填补在范围[0,1]之间； • zero - 使用零基线，即y轴. ； 另外offset 可以是一个函数；输入到该函数的是层数据；并已被标准化显示：一个二维的值数组，每个元素被表示为一个二元素的数组[x,y]；函数的返回值必须是一个表示基线y坐标的数组。例如，默认的零偏移实现如： function offset(data) { var j = -1, m = data[0].length, y0 = []; while (++j < m) y0[j] = 0; return y0; }

stack.order([order])

如果指定order，设置堆叠的排序为指定的order；如果未指定，则返回当前的order。以下字符串类型的值可被使用： • inside-out - 通过最大值的索引进行排序，然后使用平衡加权； • reverse - 反转输入层的次序； • default - 使用输入层顺序； 另外order也可以是函数；输入到该函数的是层数据，并已被标准化显示：一个二维的值数组，每个元素被表示为一个二元素的数组[x, y]；该函数的返回值必须是一个表示层顺序的索引数组。例如，默认的顺序实现如： function order(data) { return d3.range(data.length); } 参见 d3.range.

stack.x([accessor])

指定如何访问每个值位置中的x坐标。如果指定accessor，则设置访问器为指定的函数；如果未指定accessor，则返回当前绑定的访问器函数；在默认情况下假定每个输入值都有x属性：function x(d) { return d.x; } X访问器会被每一个输入值调用，并且每一层，被传递的参数有当前数据(d)和数据元素索引(i)；返回值必须是一个数字；虽然x访问器会被每层调用，堆叠布局是假定所有层的x坐标是一致的；也就是说，堆叠布局目前需要每层都是均匀的，在相同的x坐标下，都必须包含相同个数的值；如果你的数据不符合这种规则，在你进行堆叠布局前需要整理好数据符合这种规则。

stack.y([accessor])

指定如何访问每个值位置中的y坐标。如果指定accessor，则设置访问器为指定的函数；如果未指定accessor，则返回当前绑定的访问器函数；在默认情况下假定每个输入值都有y属性： function y(d) { return d.y; } X访问器会被每一个输入值调用，并且每一层，被传递的参数有当前数据(d)和数据元素索引(i)；返回值必须是一个数字；随着异常的扩大偏移，堆叠布局不会执行任何的数据自动缩放；为了简化缩放，可以使用关联的线性缩放linear scale或其他类似的。

stack.out([setter])

指定如何传递计算的基线给上层；如果指定setter，它将被用作该输出/传递功能上；如果未指定setter，则返回当前的setter；默认情况下假定每个输入值都有y和y0属性： function out(d, y0, y) { d.y0 = y0; d.y = y; } setter会被每一个输入值调用，并且每一层，被传递的参数有当前数据(d)，已计算的y0值和已计算的y厚度；在除了expand偏移的所有情况下，y厚度是相同于y的返回值，因此可被忽略。

- 魏飞译 2014.07.08.10.04
- gulu 校对 2014-12-6 23:08:08

> Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 树布局

- guluT20141102
- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

树布局能够用莱因戈尔德-蒂尔福德算法产生一个整洁的树状结点-链接图。例如，树布局可以用来组织软件的包中类的层级结构。 像其他大多数布局一样，d3.layout.tree返回的是一个对象也是一个函数。也就是说：你可以像调用其他函数一样调用布局，并且这个布局函数有可以改变它的行为的附加方法。像D3里的其他类一样，布局遵循函数的链式模式，其中setter返回布局本身，允许在简单的语句中调用多个setter方法。

d3.layout.tree()

用默认设置创建一个树布局：默认的排序是null；默认的孩子访问器假定每个输入数据是一个带有孩子数组的对象；默认的操作函数为兄弟节点指定一个结点的宽度，没有兄弟姐妹的节点指定两个结点的宽度；默认的尺寸是1X1。

tree(root)

tree.nodes(root)

运行树布局，返回一个与特殊的根结点root 有关系的结点数组。树布局是D3的层次布局家族中的一部分。这些布局遵循相似的基本结构：布局的输入参数是层次结构的根节点，输出返回值是一个经过计算的全部结点的位置的数组。 在每个结点中包含下面几个属性: •双亲-双亲结点，根节点无双亲。 •孩子-孩子结点的数组，叶子无孩子结点。 •深度-结点的深度，根节点深度为0。 •x-计算结点位置的x坐标。 •y-计算结点位置的y坐标。 尽管布局在x和y方向上有大小，这表示了一个任意的坐标系统；例如，你可以将x看做半径，将y看做角度去产生一个径向而不是笛卡尔布局。

tree.links(nodes)

给定一个特殊的结点数组nodes，比如通过nodes返回的，返回一个表示从双亲结点到每个孩子结点的对象数组。叶子结点将不再有任何链接，每个链接是一个有两个属性的对象: •源结点-双亲结点(如上文描述) •目标结点-孩子结点 这种方法是有用的，用于检索一组适合于显示的链接描述，经常与对角线形状发生器一起使用。例如： svg.selectAll("path") .data(tree.links(nodes)) .enter().append("path") .attr("d", d3.svg.diagonal());

tree.children([children])

如果指定了子节点children ，设置指定的子节点访问器函数。如果未指定子节点，返回当前子节点访问器函数，默认情况下假定输入数据是一个带有子节点数组的对象： function children(d) { return d.children; } 通常，使用d3.json很方便加载节点的层次结构，并用嵌套JSON对象代表输入层次结构。例如： { "name": "flare", "children": [{ "name": "analytics", "children": [{ "name": "cluster", "children": [{"name": "AgglomerativeCluster", "size": 3938}, {"name": "CommunityStructure", "size": 3812}, {"name": "MergeEdge", "size": 743}] }, { "name": "graph", "children": [{"name": "BetweennessCentrality", "size": 3534}, {"name": "LinkDistance", "size": 5731}] }] }] } 子节点的访问器首先在层次结构的根节点调用。如果访问器返回null，则该节点在布局遍历终止被认为是叶节点。否则，访问器应返回表示子节点数据元素的数组。

tree.separation([separation])

如果指定分割separation ，使用指定的函数计算两个相邻节点之间的分割。如果，没有指定分割，返回当前的分割函数，默认为： function separation(a, b) { return a.parent == b.parent ? 1 : 2; } 变量更适合径向布局按比例减少半径之间的分割间隙： function separation(a, b) { return (a.parent == b.parent ? 1 : 2) / a.depth; } 分割函数传入两个相邻节点a和b，必须返回节点之间期望的分割。如果布局决定放置这些相邻节点，节点通常是兄弟节点，即使节点可能是表兄弟（甚至更遥远的关系）。

tree.size([size])

如果指定尺寸size ，设置可用的布局尺寸为给指定的代表x和y两个元素的数组。如果没有指定的，返回当前的尺寸，默认是1x1。布局尺寸可能指定为x和y，但是并不限定屏幕坐标系和其他任意的坐标系统。例如，产生一个径向布局，其中breadth (x)以度为单位，depth (y)以像素为单位的半径r，例如[360,r]。 树的尺寸是tree.nodeSize特有的属性；设定tree.size设置tree.nodeSize为null。

tree.nodeSize([nodeSize])

如果指定了nodeSize，则为每个节点设置一个固定大小为代表x和y的两元素数组。如果没有指定的，则返回当前结点的尺寸，其中缺省为null表示该布局是使用整体tree.size属性，而不是使用固定的节点大小。布局尺寸用x和y指定，但是并不受限于屏幕坐标系和其他坐标系统。 nodeSize的属性是tree.size特有的；设定tree.size设置tree.nodeSize为null。

tree.sort([comparator])

如果指定比较器comparator，用指定的比较器来为布局的同级节点设置排序。如果比较器没有被指定，则返回当前组的排序，默认为空即不排序。比较函数被节点对直接调用，传入输入数据的每一个结点。默认的比较器是空，没有排序和者用树的遍历命令。例如，通过传递数据的数值来进行降序排列: function comparator(a, b) { return b.value - a.value; } 通过结点名或者关键字进行排序也是很普遍的。这可以用d3.ascending和d3.descending可以很容易实现。

tree.value([value])

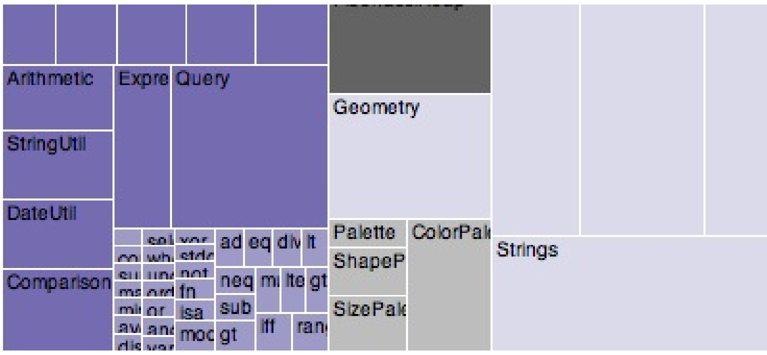
如果值是指定的，则使用指定的函数来设置值访问器。如果值没有指定，则返回默认为空的当前值访问器，意思是这个值的属性不能被计算。如果指定，值访问器被每一个输入的数据元素调用，一定能返回一个表示结点值的数字。这个值对于布局来说是没有意义的，但是是层次布局提供的通用函数。

- 阿呆不呆20141128
- gulu校对2014-12-7 10:20:53

Wiki ▸ [[API--中文手册]] ▸ [[布局]] ▸ 矩形树布局

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

矩形树图最先由Ben Shneiderman在1991年提出; 矩形树图会递归的对一块矩形区域进行切分, 以达到层级展示的效果. 正如分区布局中, 每个节点的大小都是显而易见的. 正方化的矩形树图使用近正方的矩形, 因此, 相比于传统的切块或切片图, 具有更好的可读性和节点大小易读性. 还有其他一些关于矩形树图的算法, 比如: Voronoi 和 jigsaw, 但是并不常用.



和其他 **D3** 类一样, 布局也支持链式调用.

链式调用, 即*: 一个setter方法会返回当前实例, 意味着意味着一个setter方法调用后可以紧接着另一个setter方法的调用, 如: selection.attr('x', 1).style('color', '#999')*

d3.layout.**treemap**()

使用默认的配置实例化一个新的矩形树布局: 默认的递减排序, 默认的值存取器会假定输入的数据是一个有value属性的对象, 默认的子节点存取器会假定输入的数据是一个以对象为元素的数组, 默认的绘图区域大小是[1, 1].

treemap(root)

treemap.nodes(root)

运行树形图布局，返回与指定根节点相关的节点数组。树状图布局是D3族分层布局的一部分。这些布局遵循相同的基本结构：布局的输入参数是层次结构的根节点，输出的返回值是代表所有节点计算的位置的数组。每个节点还有一些属性：

• parent -父节点，或根是null。 • children -子节点的数组，或叶节点是null。 • value -该节点的值，通过值访问器返回。 • depth -节点的深度，根开始为0。 • x -节点位置的最小横坐标。 • y -节点位置的最小纵坐标。 • dx -节点位置的x轴宽。 • dy -节点位置的y轴宽。

请注意，这会改变传递的节点！ 虽然布局有x 和 y尺寸，这表示一个任意的坐标系统；例如，你可以以x为半径y为角度产生辐射状而不是直角的布局。在直角方向，x，y，dx和dy对应于“x”，“y”，“宽度”和“高度”的SVG矩形元素的属性。

treemap.links(nodes)

给定指定的节点数组，如返回的节点，返回一个对象数组代表从父到子节点的链接。叶节点将不会有任何链接。每一个关系都有两个属性的对象：• source -父节点（如上所述）。 • target -子节点。

这种方法是有用的，用于检索一组适合于显示的链接描述，经常与对角线形状发生器一起使用。例如： svg.selectAll("path") .data(partition.links(nodes)) .enter().append("path") .attr("d", d3.svg.diagonal());

treemap.children([children])

如果指定了子节点，设置指定子节点访问器函数。如果未指定子节点，返回当前子节点访问器函数，默认情况下假定输入数据是一个对象，一个子节点数组： function children(d) { return d.children; }

通常，使用d3.json很方便加载节点的层次结构，并用嵌套JSON对象代表输入层次结构。例如： { "name": "flare", "children": [{ "name": "analytics", "children": [{ "name": "cluster", "children": [{"name": "AgglomerativeCluster", "size": 3938}, {"name": "CommunityStructure", "size": 3812}, {"name": "MergeEdge", "size": 743}] }, { "name": "graph", "children": [{"name": "BetweennessCentrality", "size": 3534}, {"name": "LinkDistance", "size": 5731}] }] }] }

子节点的访问首先在层次结构的根节点调用。如果访问器返回null，则该节点在布局遍历终止被认为是叶节点。否则，访问应返回表示子节点的数据元素的数组。

treemap.sort([comparator])

如果比较器是指定的，使用指定比较函数设置布局的兄弟节点的排序顺序。如果没有指定比较器，返回当前组的排序顺序，默认为通过相关的输入数据的数值属性降序排列： function comparator(a, b) { return b.value - a.value; }

比较器函数节点对被调用，传递每个节点的输入数据。空比较器禁用排序并使用树的遍历顺序。比较器的功能也可以用d3.ascending或d3.descending实施。

treemap.value([value])

如果指定了值value ，将值访问器为指定的函数。如果值是未指定的，则返回当前值的访问器，假定输入数据是一个带有数值属性的对象： function value(d) { return d.value; } 值访问器调用被每个输入数据元素，并且必须返回一个表示该节点的数值。这个值用于按比例设定每个节点面积的值。

treemap.size([size])

如果尺寸size 是指定的，设置现有布局尺寸，指定代表x和y的数值的两元素数组。如果尺寸不是指定的，则返回当前大小，默认为1×1。

treemap.padding([padding])

为每个树形图单元获取或设置填充，以像素为单位。填充确定保留父节点和子节点之间的额外空间量；这个空间可以用于通过外边缘表示层次结构，或保留父标签的空间。如果不使用填充，那么树中叶子的内容会完全填满布局的大小。 如果指定了填充，则设置新的填充并返回树形布局；如果没有指定填充，则返回当前的填充。填充可以指定几种方法：

• 空值禁用填充；空等同于零。 • 一个数字表示均匀填充，以像素为单位，在所有四面。 • 数组的数值表示上，右，下和左填充值。 填充也可以被指定为一个函数，它返回上面的三个值中的一个。这个函数为每个内部（非叶）节点求值，并且可以被用来动态地计算填充。

treemap.round([round])

如果四舍五入round 是指定的，设置树状图布局是否将全面取整到像素边界。这可以很好地避免SVG边缘有锯齿。如果四舍五入不是指定的，返回是否将树状图四舍五入。

treemap.sticky([sticky])

如果粘滞sticky 是指定的，设置树形布局是否是“粘滞”： 粘滞树形布局将保留整个过渡中节点的相对排列。节点分成正方化水平和垂直行，通过每行中最后一个元素存储一个z属性持续整个更新；这允许节点平滑地调整大小，没有将阻碍感知变化值的洗牌或闭塞。但请注意，这会为两种状态之一生成一个次优的布局。如果没有指定粘滞，则返回的树形布局是否已被指定粘滞。

实现注意事项：粘滞树形图的内部缓存节点数组；因此，相同的布局实例不可能再使用于多个数据集。当切换带有粘滞布局的数据集时重置缓存状态，再次调用sticky(true)。自从版本1.25.0，层次布局不再在每次调用中默认复制输入数据，因此它可能是能够消除高速缓存和使布局完全无状态。

treemap.mode([mode])

如果模式mode 是指定的，设置布局的算法。如果模式没有指定，返回当前的布局算法，默认为“squarify”。支持以下模式： • squarify - rectangular subdivision; squareness controlled via the target ratio. • slice - horizontal subdivision. • dice - vertical subdivision. • slice-dice - alternating between horizontal and vertical subdivision. • squarify -矩形细分；矩形通过目标比率控制。 • slice –水平细分。 • dice -垂直细分。 • slice-dice -水平和垂直细分之间的交替。

- ?译gulu
- 校对2014-12-7 08:43:59

> Wiki ▸ [[API--中文手册]] ▸ [[地理]]

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

参见：

- [[Paths|地理路径]] - 展示地理形状。
- [[Projections|地理投影]] - 转换地理坐标为像素坐标。
- Streams - 流几何转换。

> Wiki ▸ [[API--中文手册]] ▸ [[地理]] ▸ 地理路径

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

对于地图的可视化，D3支持少数的组件显示和操作地理数据。这些组件使用GeoJSON格式——在JavaScript中标准的地理特征表示方法。（可以参见TopoJSON格式，它是GeoJSON的扩展格式，表示的更加紧密。）。要把图形文件转换为GeoJSON，使用ogr2ogr的GDAL包的一部分。

这有一些你可能感兴趣的其他工具： TopoJSON –简化图形文件，拓扑结构和GeoJSON压缩。 Shapely –操作平面几何图形对象。 ColorBrewer –地图颜色的比例尺。 PostGIS –一个地理空间数据库。 显示地理数据最主要的机制就是用到d3.geo.path。这个类是类似于d3.svg.line和其他SVG形状生成程序：给定一个几何形状或功能的对象，它生成适合于SVG路径元素的“d”属性的路径数据串。该d3.geo.path类可以直接渲染到画布上，在动画投影时可以提供更好的性能。

d3.geo.path()

创建一个新的地理路径生成器，使用默认设置：albersUsa投影和4.5个像素点的半径。

path(feature[, index])

为给定的功能feature返回路径数据串，它可以是任何GeoJSON的特征或几何对象： Point –单个位置。 MultiPoint –一组位置。 LineString –一组位置形成一条连续的线。 MultiLineString - 位置数组的数组，形成多条线。 Polygon –位置数组的数组，形成一个多边形（可能是由小洞组成的）。 MultiPolygon –位置的多维数组，形成了多个多边形。 GeometryCollection –几何对象的数组。 Feature - 包含了几何对象其中的一个特征。 FeatureCollection –特征对象的数组。 “Sphere”类型也支持，对于绘制地球的轮廓是非常有用的。sphere球体没有坐标。一个可选的index 索引可以被指定，传递给pointRadius存取器，Index在路径生成器被selection.attr调用的时候自动地传递。 重要的是：一多边形内部是所有的点，这些点是以顺时针方向围绕着多边形。如果你的GeoJSON输入有错误顺序的多边形，那么你必须扭转它们，通过ST_ForceRHR，你也可以将你的GeoJSON转换为TopoJSON，这样的话，它就能自动生成。 显示多个功能，你可以将它们放置在一个单一的特征集和单一路径元素上： svg.append("path") .datum({type:"FeatureCollection",features:features}) .attr("d",d3.geo.path()); 另外，您也可以创建多个不同的路径元素： svg.selectAll("path") .data(features) .enter().append("path") .attr("d",d3.geo.path()); 对于集合来说利用不同的路径元素通常地比单个路径元素要慢。但是，如果你想与特征分别进行交互的话，不同的路径元素是最好的（例如，使用CSS：悬停或单击事件）。

path.projection([projection])

如果 指定了投影projection，设置为路径生成器使用的投影为指定的投影函数。如果 projection 投影没有指定，那么返回当前的投影，默认为albersUsa。projection 投影通常是D3一个内置地理投影；然而，任何函数都可以使用。投影函数采用了两元素的数字数组来代表坐标的位置，[longitude, latitude]，并返回类似的表示该投影像素位置[x, y]的两元素数字数组。例如，一个基本的spherical Mercator投影： functionmercator(coordinates){ return[coordinates[0]/360, (-180/Math.PI*Math.log(Math.tan(Math.PI/4+coordinates[1]*Math.PI/360)))/360]; } 从内在的说，这个点投影函数是包裹着一个备用的流变换，它可以执行自适应重采样。然而，备用的流不执行任何裁剪或切割。 为了更好地控制流的变换，projection 投影可以被指定为一个对象来实现流方法（参见实例：http://bl.ocks.org/mbostock/5663666）。该流的方法需要一个输出流作为输入，并返回一个投影在输入的几何体上的包装流，换句话说，它实现了projection.stream。 如果投影为null，则路径使用恒等变换，其中输入的几何体不被投影并且不是直接以原始坐标来渲染。这对已经投影的几何体快速渲染有用，或对正方形投影的快速渲染有用。

path.context([context])

如果指定了context，设置渲染上下文并返回生成的路径。如果context 为空，当在一个给定的功能调用时，路径生成器将返回一个SVG路径字符串。如果context 非空，路径生成器将替换调用函数为指定的上下文来渲染几何图形。context 必须实现以下方法： beginPath() moveTo(x, y) lineTo(x, y) arc(x, y, radius, startAngle, endAngle) closePath() 可以注意到，这是画布元素的二维渲染上下文的子集，这样画布context 可以被传递到路径生成器，在这种情况下，几何形状将直接渲染到画布上。如果context 没有指定，则返回当前渲染的上下文，默认为null。

path.area(feature)

对指定的功能feature计算投影面积（使用方形像素）。Point，MultiPoint，LineString和LineString特征有为零的区域。对多边形Polygon和多边形集合MultiPolygon的特征，该方法首先计算外部环形的面积，再减去内部有任何孔的区域。这种方法通过投影流观察进行任何的剪裁和重新采样。

path.centroid(feature)

对指定的功能feature计算投影重心（以像素为单位），这是非常方便的。比如说，标记州或国家的边界，或显示符号映射。非连续统计图，示例了围绕其质心的每个状态。这种方法观察通过投影流运行的任何剪裁和重新采样。非连续统计图：ttp://mbostock.github.com/d3/ex/cartogram.html

path.bounds(feature)

对特定的功能计算投影的边框（像素），这是非常方便的。比如说，放大到一个特定的形态。这种方法观察通过投影流运行的任何剪裁和重新采样。

path.pointRadius([radius])

如果指定半径，那么设置的半径为特定的数，用来显示点Point和多点MultiPoint的功能。如果未指定半径，则返回当前的半径。而半径通常指定为一个数字常数，它也可以被指定为对每个特征进行计算的函数，传递来自路径函数的feature 和index 参数。例如，如果你的GeoJSON数据有额外的属性，你就可以访问内置的半径函数中的属性来改变点的大小；或者，你可以用d3.svg.symbol 和一个投影来更好地控制显示。 Shape Generators

注：在D3中生成一个巨大的弧线，只需要简单地把一个LineString类型的几何对象传递给d3.geo.path。D3的投影采用great-arc插值器生成中间点（使用了自适应重采样）。因此没有必要使用形状生成器来创造大的弧线。

d3.geo.graticule

构造一个特征生成器用来创建地理刻度。

graticule()

返回一个MultiLineString几何对象表示这个刻度的所有经线和纬线。

graticule.lines()

返回LineString几何对象，用于这个刻度的每一个经线和纬线。

graticule.outline()

返回一个多边形Polygon几何对象，代表了这个地理坐标的轮廓，IE浏览器沿着它的经线和纬线界定其范围。

graticule.extent(extent)

如果extent 指定了，设置此刻度的主要和次要范围。如果没有指定范围，返回当前次要的范围，默认为⟨⟨-180°, -80° - ε⟩, ⟨180°, 80° + ε⟩⟩。

graticule.majorExtent(extent)

如果extent 指定了，设置此刻度的主要范围。如果未指定范围，返回当前主要的范围，默认为⟨⟨-180°, -90° + ε⟩, ⟨180°, 90° - ε⟩⟩。

graticule.minorExtent(extent)

如果extent 指定了，设置此刻度的较小范围。如果没有指定范围，返回当前较小的范围，默认为⟨⟨-180°, -80° - ε⟩, ⟨180°, 80° + ε⟩⟩。

graticule.step(step)

如果step 指定了，设置这个刻度的主要和次要的步长。如果没有指定step，则返回当前次要的step，默认为⟨10°, 10°⟩。

graticule.majorStep(step)

如果step 指定了，设置这个刻度的主要step。如果没有指定step，返回当前的主要step，它默认为⟨90°, 360°⟩。

graticule.minorStep(step)

如果step 指定了，设置这个刻度次要的step 。如果没有指定step，返回当前的次要step，默认为⟨10°, 10°⟩。

graticule.precision(precision)

如果指定了precision，设置这个刻度的精度，以度为单位。如果没有指定精度，则返回当前精度，默认值为2.5°。

d3.geo.circle

为在一个给定的地理位置创建带有给定半径（以度为单位）的圆中心构建一个特征生成器。

circle(arguments…)

返回一个接近圆形的GeoJSON多边形。原点访问器指定如何为给定的参数确定原点，默认的访问使用的是常量⟨0°,0°⟩。

circle.origin([origin])

如果指定了origin，设置圆的原点。一个两元坐标数组应该被指定，或访问函数。如果origin没有指定，则返回当前的原点，默认值为⟨0°,0°⟩。

circle.angle([angle])

如果指定了angle，用度数来设置圆的角半径。如果angle 没有指定，则返回当前的半径，默认为90°。

circle.precision([precision])

如果precision 指定，以度为单位设置圆形片段的精度的插值器。当一个特征被圆形裁剪时，这些内插节段就被插入了。如果没有指定precisionis，则返回当前精度，默认值为6°。
Spherical Math

d3.geo.area(feature)

返回指定功能的球形区域（以球面度为单位）。可以参考path.area，其在Cartesian plane平面上计算投影区域。

d3.geo.centroid(feature)

返回指定功能的球形重心。可参考path.centroid，其在Cartesian plane平面上计算投影重心。

d3.geo.bounds(feature)

返回指定功能的球形边框。边界框是由一个二维数组来表示：[[left, bottom], [right, top]]，其中left 是最小经度，bottom 是最小纬度，right 是最大经度和top 是最大纬度。可参考path.bounds，其在Cartesian plane平面上计算投影的边界框。

d3.geo.distance(a, b)

返回在两个点a和b之间弧度的最大距离，每个点都被指定为一个数组[longitude, latitude]，和表示十进制度的坐标。

d3.geo.length(feature)

返回弧度指定功能的大弧great-arc的长度。对于多边形，返回外环周长加上任何内置的环。

d3.geo.interpolate(a, b)

返回给定的两个位置a和b的插值器。每个位置必须被表示为一个[longitude, latitude]的二元数组。返回的插值器是一个函数，它可以接受单个参数t作为输入，其中t的范围是0～10，0返回a位置的，而1返回b的位置。中间值的插入是从a到b的大弧形。

d3.geo.rotation(rotate)

设置一个旋转为 [λ, φ, γ]形式的数组。数组的元素是以度为单位的角度，并指定按照下面的顺序来旋转：纵向，横向和原点。如果数组的最后一个元素，γ，省略了，那么这个默认值就为0。返回一个函数，该函数就如所述的那样转动到一个指定的位置。

rotation(location)

在上述描绘的顺序下，这个旋转是在一个指定的位置，根据特定的角度来旋转的。位置被指定为一个数组[longitude, latitude]，用度数表示的坐标。返回一个新的数组来表示旋转的位置。

rotation.invert(location)

该旋转是根据指定的角度和位置来旋转的，但与上述描述的相反顺序来旋转。一个位置指定为一个数组[longitude, latitude]，用度数来表示坐标。返回一个新的数组来表示旋转的位置。

- 翟琰琦译 20141124
- 何凯琳 译2014年11月27日 20:45:29
- Gulu 校对 2014-12-7 21:45:37

> Wiki ▸ [[API--中文手册]] ▸ [[地理]] ▸ 地理投影

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

D3默认包括了一些常见投影，如下所示。众多的（不太常用的）投影在扩展地理投影插件和多面体投影插件中是可用的。

由D3提供的大多数投影都是通过d3.geo.projection来创建并配置的，你可以旋转这个地球，缩放或转换画布等。除非你正在执行一个新的原始投影，否则你可能不会用D3.geo.projection来构造，但是你有可能使用这个配置方法。

d3.geo.projection(raw)

从指定的原始raw 点投影的函数，构造一个新的投影。例如，一个Mercator投影可实现为：varmercator=d3.geo.projection(function(λ,φ){ return[λ, Math.log(Math.tan(π/4+φ/2))]; });（可以参考src/geo/mercator.js全面的实现。）如果原始函数支持反转方法，则返回的投影将会显示一个对应的反转方法。 src/geo/mercator.js：ttps://github.com/mbostock/d3/blob/master/src/geo/mercator.js

projection(location)

投影从球面坐标（单位：度）转到笛卡尔坐标（单位：像素），返回数组[x, y]，给出输入数组[longitude, latitude]。如果指定的位置没有定义投影的位置，那么返回可能为null。例如，当位置在投影的裁剪边界之外的时候。

projection.invert(point)

投影反向是从直角坐标（像素），到球面坐标（度），返回一个数组[longitude, latitude]，给定输入数组[x, y]。但并非所有的投影都会实现反转，对于可逆投影，这种方法是undefined。

projection.rotate([rotation])

如果rotation 旋转指定了，设置投影的三轴旋转为指定的角度λ，φ和γ（偏航角，倾斜角和滚动角，或等效地经度，纬度和滚动），以度并返回投影。如果rotation 未指定，返回当前缺省的转动值[0, 0, 0]。如果rotation 指定且只有两个值，而不是3个值，那么滚动的角度被假设为0°。

projection.center([location])

如果center 重心指定了，则设置投影中心为指定的位置，经度和纬度度数的两元数组，并返回投影。如果没有指定中心，则返回当前的中心，默认为⟨0°,0°⟩。

projection.translate([point])

如果point 点指定了，则设置投影转变的位移为指定的二元数组[x, y]并返回的投影。如果未指定点，则返回当前变换的位移，默认为[480, 250]。变换的位移确定投影的中心像素坐标。默认转换位移的位置是⟨0°,0°⟩，在一个960×500区域的中心。

projection.scale([scale])

如果scale 比例尺指定了，则设置投影的比例尺为特定的值，并返回投影。如果未指定比例尺，返回默认的值150。比例尺因子线性地对应于投影点之间的距离。然而，比例尺因子不能一直越过投影。

projection.clipAngle(angle)

如果angle 角度指定了，则设置投影的裁剪圆半径，指定角度并返回投影。如果角度为null，切换到子午线切割，而不是小圈的裁剪。如果没有指定角度，返回当前裁剪的角度，默认为空。小圈裁剪是独立于通过clipExtent的视窗裁剪。 antimeridian cutting：http://bl.ocks.org/mbostock/3788999

projection.clipExtent(extent)

如果extent 范围指定了，则设置投影的剪辑视窗范围为指定的边界，以像素为单位并返回投影。extent 范围边界被指定为一个数组[[x0, y0], [x1,y1]]，其中x0 是视窗的左侧，y0 是顶部，x1 为右侧和y1 是底部。如果范围为null，则视窗裁剪不执行。如果extent 范围没有指定，则返回当前视窗裁剪的范围，默认为null。视窗裁剪是独立于通过clipAngle的小圈剪裁。

projection.precision(precision)

如果precision 精度指定了，则设置投影自适应重采样的临界值为指定的值，以像素为单位，并返回投影。此值对应于Douglas–Peucker 距离。如果precision 精度没有指定，则返回投影当前重采样的精度，其精度默认为Math.SQRT(1/2)。 0的精密度禁止自适应重采样。

projection.stream(listener)

返回一个投影流，包装了特定的监听器。任何几何形状的流对封装器，都是在被传输到包监听之前投影的。一个典型的投影包含多个流的变换：输入的几何形首先被转换为弧度，在三轴上旋转，在小圆圈或沿着子午线切割，最后投射到具有自适应重采样，缩放和平移的笛卡尔平面上。

d3.geo.projectionMutator(rawFactory)

构造一个新的投影，是从指定的原始点投影函数factory开始。此函数不直接返回投影，而是返回一个变换的方法，且你可以随时调用原始投影函数发生变换的方法。例如，假设你正在实施Albers equal-area conic投影，它需要配置投影的两大相似之处。使用闭包，可以实现原始投影，如下所示： // φ0 and φ1 are the two parallels functionalbersRaw(φ0,φ1){ returnfunction(λ,φ){ return[/ *compute x here* /, / *compute y here* /]; }; }

使用d3.geo.projectionMutator，可以实现一个标准的投影，使相似之处有所改变，重新分配由d3.geo.projection内部使用的原始投影： functionalbers(){ varφ0=29.5, φ1=45.5, mutate=d3.geo.projectionMutator(albersRaw), projection=mutate(φ0,φ1);

projection.parallels=function(*){ if(!arguments.length)return[φ0,φ1]; returnmutate(φ0=+*[0],φ1=+_[1]); };

returnprojection; } 因此，在创建一个可变的投影时，变化的函数永远不会暴露出来，但可以很容易地重新创建底层的原始投影。对于完全实现，可以参考src/geo/albers.js。 Standard Projections

d3.geo.albers()

d3.geo.albers是d3.geo.conicEqualArea的一个别名，默认为美国中心：scale 1000，translate [480, 250]，rotation [96°, 0°]，center ⟨-0.6°, 38.7°⟩ 和parallels [29.5°, 45.5°]，使其能够适合的显示美国，中心围绕着Hutchinson，Kansas在一个960×500的区域。中央经线和纬线是由美国地质调查局在1970年国家地图集规定的。

d3.geo.albersUsa()

埃尔伯USA投影是一个复合投影，这复合的四个埃尔伯投影的设计是用来显示美国下部48，在阿拉斯加和夏威夷旁边。虽然用于等值线图，它扩展阿拉斯加0.35x倍的区域（估计是3倍），夏威夷作为下部48显示在相同比例尺的。 埃尔伯USA投影不支持旋转或者居中。

d3.geo.azimuthalEqualArea()

方位角等面积投影也适用于等值线图。这个投影的极性方面用于联合国标志。 polar aspect : ttp://bl.ocks.org/mbostock/4364903

d3.geo.azimuthalEquidistant()

方位角的等距投影保留了从投影中心的距离，从任何投影点到投影中心的距离到大弧距离是成比例的。因此，围绕投影中心的圆圈是投影在笛卡尔平面的圆圈。这可以用于相对一个参考点的可视化距离，如可交换距离。

d3.geo.conicConformal()

兰伯特的等角的二次曲线投影投影地球形状成为一个锥形。

conicConformal.parallels([parallels])

如果parallels 指定了，则设定投影的标准平行线为特定的二元纬度数组，并返回这个投影。如果parallels 没有指定，返回当前的parallels 。

d3.geo.conicEqualArea()

埃尔伯投影，作为一个区域相等的投影，被推荐用于等值线图，因为它保留了地理特征的相对区域。

conicEqualArea.parallels([parallels])

如果parallels 指定了，设置埃尔伯投影的标准平行线为指定的二元纬度数组（以度为单位），并返回投影。如果parallels 没有指定，返回当前parallels 。为了最大限度地减少失真，平行线应选择为围绕投影的中心。

d3.geo.conicEquidistant()

conicEquidistant.parallels([parallels])

如果parallels 指定了，设定投影的标准parallels 到指定的二元纬度数组（度），并返回投影。如果parallels 没有指定，返回当前的parallels 。

d3.geo.equirectangular()

这个正方形投影，或plate carrée投影，是最简单可行的地理投影：标识函数。它既不等面积也不等角，但有时用于光栅数据。见光栅重投影的一个例子，源图像使用正方形投影。 raster reprojection：ttp://bl.ocks.org/mbostock/4329423

d3.geo.gnomonic()

球心投影是一个方位角投影，它投射大圆为直线。参考interactive gnomonic 的例子。 interactive gnomonic：ttp://bl.ocks.org/mbostock/3795048

d3.geo.mercator()

这个球面的Mercator投影是常用的分片式映射库（例如OpenLayers 和Leaflet）。例如显示栅格分片与Mercator投影，可以参见d3.geo.tile插件，它是正形投影的，然而，它的推行造成了世界范围地区严重失真，因此不建议使用choropleths。

d3.geo.orthographic()

正投影是适合于显示单个半球的方位投影，透视的点在无穷远处。可以看世界观光的动画和互动正投影的例子。对于一般的透视投影，可以参考卫星投影。

d3.geo.stereographic()

立体投影是另一个角度（方位）投影。在表面上看，透视的点是球体。因此，这是常用的天体图。参考交互式立体为例。

d3.geo.transverseMercator()

横向的墨卡托投影。 Raw Projections

D3提供了几个原始的投影，当一个复合投影实现时，设计重用（如Sinu–Mollweide，它结合了原始的正弦曲线和摩尔维特投影）。原始投影在使用之前，通常是用d3.geo.projection封装着。这些点函数都是采用球面坐标λ和φ（弧度）作为输入，并返回一个二元数组（也是用弧度）作为输出。许多原始投影从平面坐标映射到球面坐标，实现了一个反转的投影。

d3.geo.albers.raw(φ0, φ1)

是d3.geo.conicEqualArea.raw的一个别名。

d3.geo.azimuthalEqualArea.raw

原始方位角的等面积投影。

d3.geo.azimuthalEquidistant.raw

原始的等距方位投影。

d3.geo.conicConformal.raw(φ0, φ1)

返回一个原始的等角的二次曲线投影，并用弧度指定parallels。

d3.geo.conicEqualArea.raw(φ0, φ1)

返回一个原始的埃尔伯投影，并用弧度指定parallels。

d3.geo.conicEquidistant.raw(φ0, φ1)

返回一个原始等距圆锥投影，并用弧度指定parallels。

d3.geo.equirectangular.raw

原始的正方形投影。

d3.geo.gnomonic.raw

原始的球心投影。

d3.geo.mercator.raw

原始的墨卡托投影。

d3.geo.orthographic.raw

原始的正投影。

d3.geo.stereographic.raw

原始的立体投影。

- 何凯琳译20141129
- gulu校对2014-12-7 23:27:06

> Wiki ▸ [[API Reference]] ▸ [[Geo]] ▸ **Geo Streams**

For fast transformations of geometry without temporary copies of geometry objects, D3 uses **geometry streams**. The main d3.geo.stream method converts a GeoJSON input object to a stream: a series of method calls on a *stream listener*. In addition, D3 provides several stream transformations that wrap listeners and transform the geometry. For example, the projection.stream interface transforms spherical coordinates to Cartesian coordinates, and d3.geo.path serializes geometry to either SVG or Canvas. Internally, clipping and rotating are also implemented as stream transformations.

d3.geo.**stream**(*object*, *listener*)

Streams the specified GeoJSON *object* to the specified stream *listener*. (Despite the name "stream", these method calls are currently synchronous.) While both features and geometry objects are supported as input, the stream interface only describes the geometry, and thus additional feature properties are not visible to listeners.

Stream Listeners

Stream listeners must implement several methods to traverse geometry. Listeners are inherently stateful; the meaning of a point depends on whether the point is inside of a line, and likewise a line is distinguished from a ring by a polygon.

listener.**point**(*x*, *y*[, *z*])

Indicates a point with the specified coordinates *x* and *y* (and optionally *z*). The coordinate system is unspecified and implementation-dependent; for example, projection streams require spherical coordinates in degrees as input. Outside the context of a polygon or line, a point indicates a point geometry object (Point or MultiPoint). Within a line or polygon ring, the point indicates a control point.

listener.**lineStart**()

Indicates the start of a line or ring. Within a polygon, indicates the start of a ring. The first ring of a polygon is the exterior ring, and is typically clockwise. Any subsequent rings indicate holes in the polygon, and are typically counterclockwise.

listener.**lineEnd**()

Indicates the end of a line or ring. Within a polygon, indicates the end of a ring. Unlike GeoJSON, the redundant closing coordinate of a ring is *not* indicated via point, and instead is implied via lineEnd within a polygon. Thus, the given polygon input:

```
{
  "type": "Polygon",
  "coordinates": [
    [[0, 0], [1, 0], [1, 1], [0, 1], [0, 0]]
  ]
}
```

Will produce the following series of method calls on the listener:

```
listener.polygonStart();
listener.lineStart();
listener.point(0, 0);
listener.point(1, 0);
listener.point(1, 1);
listener.point(0, 1);
listener.lineEnd();
listener.polygonEnd();
```

listener.**polygonStart**()

Indicates the start of a polygon. The first line of a polygon indicates the exterior ring, and any subsequent lines indicate interior holes.

listener.**polygonEnd**()

Indicates the end of a polygon.

listener.**sphere**()

Indicates the sphere (the globe; the unit sphere centered at ⟨0,0,0⟩).

Stream Transforms

A stream transform wraps a stream listener, transforming the geometry before passing it along to the wrapped listener. A geographic projection is one example of a stream transform. The d3.geo.transform class provides an easy way of implementing a custom stream transform.

d3.geo.**transform**(*methods*)

Creates a new stream transform using the specified hash of methods. The hash may contain implementations of any of the standard stream listener methods: sphere, point, lineStart, lineEnd, polygonStart and polygonEnd. Any method that is *not* present in the specified hash will be implemented a pass-through directly to the wrapped stream. To access the wrapped stream within a method, use `this.stream`. For example, to implement a simple 2D matrix transform:

```
function matrix(a, b, c, d, tx, ty) {
  return d3.geo.transform({
    point: function(x, y) { this.stream.point(a * x + b * y + tx, c * x + d * y + ty); },
  });
}
```

This transform can then be used in conjunction with d3.geo.path. For example, to implement a 2D affine transform that flips the *y*-axis:

```
var path = d3.geo.path()
    .projection(matrix(1, 0, 0, -1, 0, height));
```

transform.**stream**(*listener*)

Given the specified stream *listener*, returns a wrapped stream listener that applies this transform to any input geometry before streaming it to the wrapped listener.

d3.geo.**clipExtent**()

Create a new stream transform that implements axis-aligned rectangle clipping. This is typically used to clip geometry to the viewport after projecting.

clipExtent.**extent**([*extent*])

If *extent* is specified, sets the clip extent to the specified rectangle [[*x0*, *y0*], [*x1*, *y1*]] and returns this transform. If *extent* is not specified, returns the current clip extent, which defaults to [[0, 0], [960, 500]].

> Wiki ▸ [[API--中文手册]] ▸ 几何

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

- [[Voronoi|泰森多边形]] - 由一组点计算Voronoi密铺或Delaunay三角剖分 。
- [[Quadtree|四叉树]] - 由一组点计算四叉树。
- [[Hull|赫尔]] - 由一组点计算凸包。
- [[Polygon|多边形]] - 多边形计算，例如求面积、剪裁等。

> Wiki ▸ [[API--中文手册]] ▸ [[几何]] ▸ 泰森多边形

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

Voronoi布局对于无形的交互地区尤其有用，在Nate Vack's Voronoi picking例子中被证实，看Tovi Grossman's关于 bubble cursors的论文，以了解相关内容。 Voronoi picking：http://bl.ocks.org/njvack/1405439

d3.geom.()

创建一个带默认访问器的Voronoi布局。

voronoi(data)

返回多边形的数组，用于指定的数据数组中的每个输入顶点。如果有任何顶点重合或具有NaN的位置，这个方法的行为是undefined：最有可能的，将返回无效的多边形！你应该在计算曲面细分之前过滤无效的顶点，合并重合的顶点。

voronoi.x([x])

如果x是指定了的，则设置x坐标访问器。如果x坐标没被指定，返回当前的x坐标访问器。默认为： function(d){returnd[0];}

voronoi.y([y])

如果y是指定了的，则设置y坐标访问器。如果y坐标没被指定，返回当前的y坐标访问器。默认为： function(d){returnd[1];}

voronoi.clipExtent([extent])

如果范围是指定的，设置Voromoi布局的剪切范围为指定的范围并返回这个布局。范围边界被指定为一个数组[[x0, y0], [x1, y1]]，其中X0是范围的左侧，Y0是顶部， x1为右侧和Y1是底部。如果范围为空，剪切不执行。如果没有指定范围，则返回当前剪切的范围，默认为空。

看这个例子：http://bl.ocks.org/mbostock/4237768，剪切范围的使用被强烈推荐，因为没剪切的多边形可能有很大部分坐标没有被正确显示。

另外，你也可以使用自定义的没有指定大小的剪裁，无论是在SVG或者通过 polygon.clip后续处理。

voronoi.links(data)

返回指定数据数组的Delaunay三角作为links的数组。每一个link都有下面的属性： Source-源节点（数据中的一个元素）• target-目标节点（数据中的一个元素） 案例力导向美国地图：http://bl.ocks.org/mbostock/1073373使用这样的链接数组创建了一个力导向的地图。

voronoi.triangles(data)

返回指定的数据数组的Delaunay三角作为三角形的阵列。每个三角形是包含data中的元素的一个三元素数组。

- 谁浮T20141125
- guluP2014-12-8 20:55:17

> Wiki ▸ [[API--中文手册]] ▸ [[几何]] ▸ 四叉树

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

四叉树是一个二维空间递归细分。它使用了平分划分实现，将每一个块平分成了四个同等大小的方块。每个点存在于一个唯一的节点中；如果同一位置包含多个点，那么这多个点中的其中一些点将存储于内部节点，而非叶子结点。四叉树可用于加速各种空间操作，例如计算正多边形的体积的Barnes-Hut近似算法或者冲突检验。

d3.geom.quadtree()

创建一个新的四叉树工厂使用默认的x访问器，y访问器及范围。返回的函数可以用来从带有工厂的配置的数据创建任意个四叉树。

quadtree(points)

为指定的点数据数组构造一个新的四叉树，返回新四叉树的根结点。每一个点的X和Y坐标使用当前x及y访问器函数确定。通过增量地添加点来建立四叉树，指定的点数组可以为空，之后点可以随后添加到返回的根节点；在这种情况下，你必须指定四叉树的范围。 四叉树的每一个结点都有多个属性。 nodes——按顺序排列四个子结点的稀疏数组：top-left, top-right, bottom-left, bottom-right。 leaf ——内部结点与叶子结点的布尔表示。 point ——这个点关联的节点，如果有的话（可能适用于内部或叶节点）。 X——关联点的x坐标，如果有的话。 Y——关联点的y坐标，如果有的话。 返回的根结点也可定义了增加方法add 或者访问方法visit 。

root.add(point)

为四叉树增加指定的新点。

root.visit(callback)

访问四叉树的每个结点，利用参数 {node, x1,y1, x2, y2} 为每个结点调用指定的回调函数，结点按先序遍历。若调用给定结点的回调函数返回值为真，则表示不能访问此结点的子结点，否则表示所有子结点均能被访问。

quadtree.x([x])

若设置了x参数，则为四叉树设置横坐标访问器，并返回四叉树。反之，则返回当前默认的横坐标访问器。 function(d) { return d[0]; } 对每一个加入到四叉树中的点，不管是初始化构造的还是后面新增的，都可以通过参数 {d, i}调用x访问器，d表示当前点，i表示所有点数组的索引。X访问器必须返回一个数值，表明给定点的x坐标。如果需要得花，x访问器也可以被定义为一个常数，而非一个函数。

quadtree.y([y])

若设置了y参数，则为四叉树设置纵坐标访问器，并返回四叉树。反之，则返回当前默认的纵坐标访问器 function(d) { return d[1]; } 对每一个加入到四叉树中的点，不管是初始化构造的还是后面新增的，都可以通过参数 {d, i}调用y访问器，d表示当前点，i表示所有点数组的索引。y访问器必须返回一个数值，表明给定点的y坐标。如果需要得花，y访问器也可以被定义为一个常数，而非一个函数。

quadtree.extent([extent])

若设置了extent参数，则为四叉树设置范围，然后返回四叉树。反之，则返回当前范围，其默认为空。 当范围为空时，范围将会自动扫描输入点数组自动计算并传到四叉树构造器。否则，范围必须用二维数组[[x0, y0], [x1,y1]]明确定义，x0和y0为范围的下限，x1和y1为范围的上限。当从最初为空的节点慢慢构建一个四叉树，设置范围是非常必要的。 曼妙征程译20141127咕噜校对2014-12-8 20:24:31

> Wiki ▸ [[API--中文手册]] ▸ [[几何]] ▸ 多边形

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

d3.geom.polygon(vertices)

返回顶点的输入数组，并且附有一些其他方法，如下面所描述

polygon.area()

返回此多边形的标定区域。如果顶点是逆时针顺序，面积为正，否则为负。

polygon.centroid()

返回一个表示此多边形的质心的两元素数组。

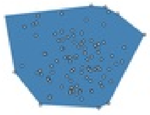
polygon.clip(subject)

对这个多边形剪切主题多边形。换句话说，返回一个多边形表示这个多边形和主题多边形的交集。假定剪切的多边形是逆时针方向以及凸多边形。

- 谁浮T20141125
- guluP20141208 2014年12月8日 21:07:33

> Wiki ▸ [[API Reference]] ▸ [[Geometry]] ▸ **Hull Geom**

d3.geom.**hull**()



Create a new hull layout with the default *x*- and *y*-accessors.

hull(*vertices*)

Returns the convex hull for the specified *vertices* array, using the current x- and y-coordinate accessors. The returned convex hull is represented as an array containing a subset of the input vertices, arranged in counterclockwise order (for consistency with polygon.clip).

Assumes the *vertices* array is greater than three in length. If *vertices* is of length <= 3, returns [].

hull.**x**([*x*])

If *x* is specified, sets the x-coordinate accessor. If *x* is not specified, returns the current x-coordinate accessor, which defaults to:

```
function(d) { return d[0]; }
```

hull.**y**([*y*])

If *y* is specified, sets the y-coordinate accessor. If *y* is not specified, returns the current y-coordinate accessor, which defaults to:

```
function(d) { return d[1]; }
```

> Wiki ▸ [[API--中文手册]] ▸ 行为

参见：

- [[Drag|拖动]] — 拖动。
- [[Zoom|缩放]] — 缩放。

> Wiki ▸ [[API--中文手册]] ▸ [[行为]] ▸ 拖动

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

该行为会自动创建事件监听器来处理元素的拖动，可支持鼠标事件和触摸事件。

d3.behavior.drag()

构造一个新的拖拽行为；

drag.on(type[, listener])

注册指定的监听器listener 来接收拖动行为中指定类型type的事件；如果监听器listener 未指定，则为指定的类型type的事件返回当前已注册的监听器（事件类型可能包含命名空间，查看详细参见： dispatch.on），可支持的事件类型包括： dragstart - 拖动开始时； drag - 拖动移动时； dragend - 拖动结束时（放下时）；拖动事件（除了dragstart 和dragend）使用x和y属性来表示本地坐标系中拖动行为当前位置；默认情况下，这个位置即是鼠标（或触摸）的位置，然而，这个位置可以通过指定一个原点修改；拖动事件同时也提供dx和dy属性来表示鼠标的“瞬时”偏移量（相对于上一时刻的偏移：正数或负数），这两个属性有时比指定一个明确地原点更方便。

在拖动的动作中，一些浏览器的默认动作会自动被阻止（如：选择文本），另外，点击事件的默认行为随即变成一个非空拖动动作也会被禁止，以便允许链接的拖动。当在可拖动元素中注册了点击事件，你可以这样来检查click事件是否被禁止： selection.on("click", function() { if (d3.event.defaultPrevented) return; // click suppressed console.log("clicked!"); }); 当拖拽事件结合其他事件一起使用时，你可以考虑阻止源事件，来避免多个动作的发生：drag.on("dragstart", function() { d3.event.sourceEvent.stopPropagation(); // silence other listeners });

drag.origin([origin])

如果指定了原点origin， 设置原点访问器为指定的函数；如果未指定原点origin， 则返回当前原点访问器，原点默认为null；

原点访问器函数被用来确定拖拽开始时的起始位置，这就允许拖拽行为保存鼠标位置相对于开始元素位置的偏移量；如果原点访问器为空，元素位置被设置为鼠标位置时可能会有明显的跳动；如果指定了原点访问器，该函数会在鼠标被按下时调用，该函数会像其他函数调用

方式一样被调用，即：会传递当前元素的数据d和元素索引i，还有this上下文代表当前点击的DOM元素；要访问当前发生的事件，可以使用d3.event；指定的原点访问器必须返回一个包含被拖动元素开始坐标x和y的对象；

原点访问器常常指定为恒等函数：function(d) { return d; }，当前元素已绑定了一个包含坐标位置x和y的对象时是最合适的，详细参见：http://bl.ocks.org/1557377。

- 魏飞T20141125
- guluP20141208 2014-12-8 21:53:51

> Wiki ▸ [[API--中文手册]] ▸ [[行为]] ▸ 缩放

- 如发现翻译不当或有其他问题可以通过以下方式联系译者:
- 邮箱：zhang_tianxu@sina.com
- QQ群：D3数据可视化205076374，大数据可视化436442115

该行为会自动在容器元素中创建事件监听器来处理元素的缩放和平移动作，可支持鼠标事件和触摸事件。

d3.behavior.zoom()

构造一个新的缩放行为。

zoom(selection)

应用缩放行为到指定的选择器selection，注册所需的事件监听器，支持缩放和拖拽行为。

zoom.translate([translate])

指定当前的缩放平移向量为translate；如果未指定translate，返回当前平移向量，默认：[0, 0]。

zoom.scale([scale])

指定当前的缩放比例，如果未指定scale，则返回当前的缩放比例，默认为1。

zoom.scaleExtent([extent])

指定缩放比例的允许范围为一个包含两个数值元素的数组，[minimum, maximum]；如果未指定extent，则返回当前的比例范围，默认为：[0, Infinity]。

zoom.center([center])

如果指定了center，为鼠标滚轮设置缩放的焦点（focal point [x, y]）为center，并返回当前的缩放行为对象；如果未指定center，则返回当前的缩放焦点，默认为null；一个为null的缩放焦点即：是以当前鼠标指针为缩放焦点。

zoom.size([size])

如果指定了size，设置视窗大小为指定的尺寸size（[width, height]），并返回当前的缩放行为对象；如果未指定size，返回当前的视窗大小，默认为：[960, 500]；size是支持在平移时平滑的进行缩放行为。

zoom.x([x])

指定x比例，并且它的定义域会在缩放时自动调整；如果未指定[x]，返回当前的x缩放，默认为null；如果缩放的定义域或范围是通过函数修改的，则这个函数应该被再次调用。设置x比例，并重置缩放比例为1，平移为[0, 0]。

zoom.y([y])

指定y比例，并且它的定义域会在缩放时自动调整；如果未指定[y]，返回当前的y缩放，默认为null；如果缩放的定义域或范围是通过函数修改的，则这个函数应该被再次调用，设置y比例，并重置缩放比例为1，平移为[0, 0]。

zoom.on(type, listener)

注册监听器listener来接收指定事件类型type的缩放行为；支持的事件类型有： zoomstart - 缩放开始时 (e.g., touchstart)； zoom - 缩放行为发生时 (e.g., touchmove)； zoomend - 缩放行为结束时 (e.g., touchend)； 如果相同的事件类型已经指定了监听器，则首先会移除老的监听器，在加入新的监听器；要给一个事件类型注册多个监听器，则可以使用事件类型加命名空间后缀的形式，如："zoom.foo" 和 "zoom.bar"；想要移除监听器，只需要传入null即可； 对于鼠标滚轮事件，针对浏览器而言是没有明确的结束和开始通知的，在50毫秒内的多个事件会被分组到单个缩放中；如果想要更好的处理该种滚轮类型，请参考您的浏览器厂商，来获得更好的支持； 事件发生时，d3.event中会包含以下属性： scale - 一个数值，即当前的比例； translate - 一个含有两个数值元素的数组，代表平移向量。

zoom.event(selection)

如果selection是一个选择器，立刻给注册的监听器分派一个缩放动作，作为三个时间序列：zoomstart，zoom 和 zoomend；这可以在设置了平移或比例后立刻触发监听器；如果selection是一个平移（transition），注册适当补间，使得缩放行为在过渡的过程中分派事

件：zoomstart 事件发生在此前设定的视图平移开始时，zoom 事件则时刻发生在平移transition中，zoomend 事件则发生在平移transition结束时；注意：如果用户开始一个新的缩放行为前，之前的缩放行为动画会被立刻中断。

- 魏飞T20141125
- guluP20141208 2014-12-8 22:21:03